Ecole Polytechnique Fédérale de Lausanne
Faculté Sciences de Base
Chaire de Recherche Opérationnelle Sud Est

Projet de master 2005-2006

Etude et illustration de méthodes itératives d'optimisation non linéaire

Travail présenté par : Thierno Diallo (section d'informatique)

Enseignant responsable : Professeur Dominique de Werra
Assistant responsable : Benjamin Leroy-Beaulieu

Lausanne, avril 2006

Prof. D. de Werra

Chaire de Recherche Opérationnelle Sud-Est
EPFL, FSB - IMA - ROSE
1015 Lausanne

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Hiver 05/06

Travail pratique de Master

Sujet :	Etude et illustrations de méthodes itératives d'optimisation non linéaire
Candidat :	Thierno Diallo, section d'informatique
Responsable :	Benjamin Leroy-Beaulieu

Introduction

En raison de leurs applications nombreuses, on a besoin de logiciels de plus en plus performants pour résoudre des problèmes d'optimisation non linéaire de grande taille.

Des techniques variées ont été proposées pour s'attaquer à des problèmes comportant des fonctions objectifs et des contraintes essentiellement non linéaires.

But du projet

Il s'agira d'étudier un certain nombre de méthodes classiques basées notamment sur les gradients des fonctions à optimiser ; ces techniques seront analysées du point de vue de leur convergence et le candidat aura pour tâche de les mettre en oeuvre en les programmant, de les comparer et d'élaborer un outil graphique permettant d'illustrer le fonctionnement des méthodes itératives de manière didactique.

Il fournira enfin sous une forme aussi compacte que possible des précisions sur les champs d'application de ces techniques et sur leurs qualités respectives.

Rapport et présentation orale

Le candidat suivra les indications du professeur et des collaborateurs responsables et les mettra au courant de l'avancement du projet **au moins une fois par semaine**. Une présentation intermédiaire du travail sera fixée ultérieurement.

Chaque phase du projet sera détaillée dans un rapport à remettre en 4 exemplaires

le **vendredi 24 février 2006** à midi au plus tard. Le rapport contiendra les points suivants :

1. la présente donnée du sujet ;
2. une introduction didactique et motivée du travail ;
3. une explication détaillée des résultats mis en évidence ainsi que leurs intérêts.
4. les performances obtenues par les méthodes développées avec interprétation des résultats et comparaison avec d'autres méthodes le cas échéant.
5. des suggestions pour une extension et un approfondissement du sujet.
6. une bibliographie (avec des références précises).
7. un CD ou une disquette contenant la version électronique du rapport, les sources LaTeX, ainsi que les codes sources des programmes développés.

Références

– D.P. Bertsekas, *Nonlinear programming*, Athena Scientific, 1999
– D. Luenberger, *Introduction to linear and nonlinear programming*, Addison-Wesley, 1973
– M. Minoux, *Programmation mathématique*, (2 tomes), Dunod 1983.

Addendum

Parmi les méthodes à illustrer, on compterea les méthodes de gradient et leurs variations (gradient réduit, gradient conjugué), les méthodes de Newton et leurs variations ainsi que les méthodes de pénalités (intérieures et extérieures) et celles de plans sécants.

Il s'agira de donner aussi des techniques fournissant des solutions admissibles initiales.

Table des matières

Chapitre 1

Introduction

Les problèmes d'optimisation sont incontournables et sont rencontrés dans tous les domaines des sciences de l'ingénieur. A cause, entre autres, de la concurrence qui n'a eu de cesse de se développer entre les différents acteurs du marché, les chercheurs ne purent pas se contenter bien longtemps de solutions uniquement vouées à satisfaire les contraintes. Pour ces raisons, le développement des modèles théoriques et des techniques traitant des problèmes d'optimisation a connu une accélération spectaculaire, particulièrement après la deuxième guerre mondiale.

Durant cette période, les ingénieurs se sont trouvés face à des problèmes à la taille et à la complexité croissante, ce qui fut une motivation pour la recherche de méthodes de résolution fiables et systématiques. La plupart d'entre elles reposent sur un socle solide de résultats théoriques établissant les conditions pour leur convergence vers la solution optimale recherchée.

Dans cette optique, nous devons pouvoir disposer de logiciels capables d'appliquer ces méthodes a des problèmes de taille variable et d'en extraire les résultats, ce qui a donné naissance à ce projet de diplôme. Celui-ci se concentre essentiellement sur les problèmes d'optimisation dont les fonctions objectif et les contraintes sont non linéaires. Il a donc pour but principal l'implémentation dans un langage de programmation, en l'occurence Java, d'algorithmes d'optimisation non linéaires, ce qui nous permettra ensuite de les comparer, d'étudier le comportement en pratique des méthodes au-delà des considérations purement théoriques, et d'établir ainsi plus précisément, à la lumière de leur utilisation pratique, leurs champs d'application et de mettre en évidence leurs qualités et leurs défauts respectifs.

Le logiciel développé au cours de ce projet a également pour vocation de permettre l'illustration et la démonstration de l'exécution des méthodes d'une manière aussi didactique que possible, afin d'en faciliter et d'en rendre plus intuitive la compréhension. Le présent document contiendra ainsi, après un bref exposé des résultats théoriques et des définitions importants, la description et l'étude comparative des méthodes implémentées. Puis, les question plus particulièrement relatives a l'application elle-même seront abordées.

1.1 Rappel de quelques définitions

Nous allons ici rappeler brièvement les définitions de quelques notions importantes auxquelles nous ferons appel par la suite, ainsi que quelques propriétés.

Pour d'évidentes raisons de place, ce document ne peut prétendre présenter les démonstrations de tous les théorèmes mentionnés. Le lecteur désireux de consulter leurs preuves pourra se reporter aux ouvrages de référence dont la liste se trouve à la fin de ce rapport. Au cours de celui-ci, les théorèmes seront énoncés ou rappelés sans entrer dans les détails de leur démonstration.

1.1.1 Ensembles et fonctions convexes

Définition 1.1.1 (**ensemble convexe**)**.** Soit l'ensemble $C \subseteq \mathbb{R}^n$. C est *convexe* si

$$\lambda x + (1-\lambda)y \in C, \qquad \forall x, y \in C,\ \forall \lambda \in [0,1].$$

Définition 1.1.2 (**fonction convexe**)**.** Soit l'ensemble convexe $C \subseteq \mathbb{R}^n$. Une fonction $f : C \mapsto \mathbb{R}$ est *convexe* si

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y), \qquad \forall x, y \in C,\ \forall \lambda \in [0,1].$$

Une fonction f est *concave* si $-f$ est convexe. Une fonction f est *strictement convexe* si l'inégalité ci-dessus est stricte pour tout $x, y \in C$ tels que $x \neq y$ et tout $\lambda \in (0,1)$.

Définition 1.1.3 (**hyperplan**)**.** Un *hyperplan* H est un ensemble de la forme $\{x \mid a \cdot x = b\}$ où $x, c \in \mathbb{R}^n$, $b \in \mathbb{R}$ et $\cdot$ désigne le produit scalaire de deux vecteurs.

Propriété 1.1.1. Si $C_1, ..., C_r$ sont des ensembles convexes, alors l'intersection $\bigcap_{i=1}^{r} C_i$ est convexe.

1.2 Le problème

Il existe de nombreuses sortes de problèmes d'optimisation. Certaines caractéristiques permettent de les distinguer : comportent-ils des contraintes ? Les fonction en jeu sont-elles linéaires ? Sont-elles quadratiques ? Sont-elles convexes ? Les domaines de définition des fonctions sont-ils continus ou discrets ? Tous ces problèmes possèdent des structures différentes et ne peuvent être traités de la même façon. Le présent projet a pour vocation de se focaliser sur les problèmes d'optimisation continue et non linéaire. Ceux-ci sont habituellement définis par un ensemble X appelé *ensemble de solutions admissibles* et une *fonction objectif* $f : X \mapsto \mathbb{R}$ qui associe à chacun des éléments de X un nombre réel, ou un *coût* (nous désignerons d'ailleurs parfois également cette fonction par l'appellation *fonction de coût* ; au sujet de X, la dénomination d'*ensemble admissible* ou encore de *domaine admissible* sera utilisée tantôt, dans un souci d'abréviation). Nous considérons le cas où X est un sous-ensemble de $\mathbb{R}^n$. Ses éléments sont donc des vecteurs de nombres réels à n dimensions de la forme $(x_1, x_2, ..., x_n)$, avec $x_i \in \mathbb{R}\ \forall i \in \{1, 2, ..., n\}$. Le problème consiste à trouver l'élément de X dont le coût est minimal (ou maximal, mais cela ne change rien à la difficulté ou aux types de méthodes employées). Cela peut être formulé de manière plus concise comme suit :

$$\begin{aligned} &\text{minimiser } f(x) \\ &\text{sous contrainte } x \in X \end{aligned}$$

X est généralement défini par une collection de contraintes exprimées sous forme d'égalités et d'inégalités. Nous admettons que f est continue et continuellement différentiable, et parfois même nous admettrons que f est deux fois continuellement différentiable. Les dérivées seconde et première jouent un rôle important aussi bien dans la caractérisation des solutions optimales que dans les idées qui conduisent à la plupart algorithmes utilisés et implémentés au cours de ce projet.

Par optimisation sans contraintes, nous désignons le cas particulier où $X = \mathbb{R}^n$. Parfois, les algorithmes de résolution sont fondés sur les mêmes principes lorsque le problème est contraint

ou lorsqu'il ne l'est pas. Ils nécessitent seulement d'être adaptés afin de produire des résultats convenables dans les deux cas, conduisant à des algorithmes différents sans que les idées fondamentales sur lesquelles ils se basent le soient foncièrement. En d'autres termes, certains algorithmes pour l'optimisation non linéaire avec des contraintes sont dérivés des algorithmes d'optimisation non linéaire sans contraintes, avec quelques modifications destinées à prendre en charges ces dernières (notamment à ne pas sortir de X).

Néanmoins, il existe aussi des algorithmes ayant été développés spécifiquement pour chacun des deux cas et dont l'idée principale repose sur la structure du problème, que l'ensemble des solutions admissibles se généralise à $\mathbb{R}^n$ ou qu'il soit restreint à n'en être qu'un sous-ensemble. Dans la section suivante nous donnons la définition formelle de la notion d'optimum et de la notion d'algorithme itératif qui est fondamentale car se sont de tels algorithmes qui ont été implémentés et qui nous permettront de résoudre les problèmes.

1.3 Minima locaux et globaux

Soient l'ensemble $S \subseteq \mathbb{R}^n$ et une fonction $f : S \mapsto \mathbb{R}$. Les minima locaux et globaux de f sur S sont définis de la manière suivante :

Définition 1.3.1 (**minimum local**)**.** Intuitivement, un vecteur $x^* \in S$ est un *minimum local* de f sur S s'il a un coût plus faible que celui de ses voisins. Formellement, x^* est un minimum local de f sur S si $\exists\ \epsilon > 0$ tel que

$$f(x^*) \leq f(x) \qquad \forall x \in S \text{ avec } |x - x^*| < \epsilon$$

où $|v|$ désigne la norme du vecteur v.

Le minimum local est *strict* si

$$f(x^*) < f(x) \qquad \forall x \in S \text{ avec } |x - x^*| < \epsilon.$$

Définition 1.3.2 (**minimum global**)**.** Un vecteur $x^* \in S$ est un *minimum global* de f sur S s'il a un coût plus faible que celui de tous les autres vecteurs dans S. Formellement, x^* est un minimum global de f sur S si

$$f(x^*) \leq f(x) \qquad \forall x \in S.$$

Le minimum global est *strict* si

$$f(x^*) < f(x) \qquad \forall x \in S.$$

Les maxima locaux et globaux sont définis de manière similaire. Notons que x^* est un maximum local (respectivement global) de la fonction f sur l'ensemble S si x^* est un minimum local (respectivement global) de $-f$ sur S. Il découle de cette observation que tout problème de maximisation peut être réduit immédiatement à un problème de minimisation (et inversement) en multipliant la fonction objectif par -1.

Remarque : dans le cas d'une fonction objectif convexe, il n'y a pas de distinction entre minimum local et global : tout minimum local est également global, comme l'établit le thèorème suivant.

Théorème 1.3.1. Soit $f : X \subseteq \mathbb{R}^n \mapsto \mathbb{R}$ une fonction convexe définie sur l'ensemble convexe X. Alors, tout minimum local de f sur X est également un minimum global. Si f est strictement convexe, alors il existe au plus un minimum global de f.

1.4 La notion d'algorithme itératif

Considérons le problème consistant à minimiser $f(x)$ sous la contrainte $x \in X$. Un *algorithme itératif* permettant de résoudre ce problème est un processus itératif générant une suite de vecteurs $x^0, x^1, ..., x^n$ de X en fonction d'une séquence d'instructions et d'une condition d'arrêt. La production d'un vecteur x^{k+1} à partir d'un vecteur x^k (qui tous deux appartiennent à X) constitue une *itération* de l'algorithme. Un tel algorithme est dit *de descente*, si le coût du vecteur généré à l'itération $k+1$ est stictement inférieur au coût du vecteur généré à l'itération k, c'est-à-dire si

$$f(x^{k+1}) < f(x^k) \qquad \forall k \geq 0.$$

Etant donné un vecteur x^k, et en appliquant la séquence d'instructions de l'algorithme, nous obtenons un nouveau vecteur x^{k+1}. Ce processus peut être décrit formellement en définissant la notion suivante.

Définition 1.4.1 (**formelle d'un algorithme**)**.** La *formelle d'un algorithme* est une application $A : X \subseteq \mathbb{R}^n \to Y \subseteq P(X)$. Le vecteur x^{k+1} est donc généré par l'algorithme de la manière suivante en fonction de x^k : A est appliquée à x^k et l'on choisit $x^{k+1} \in A(x^k) \subseteq P(X)$.

Donnons également la définition de la notion utile de limite d'une séquence de points $\{x^k\}$:

Définition 1.4.2 (**point limite**)**.** L'on appelle *point limite* d'une séquence $\{x^k\}$ de points de $\mathbb{R}^n$ tout point $x \in \mathbb{R}^n$ tel qu'il existe une sous-séquence de $\{x^k\}$ qui converge vers x.

Chapitre 2

Conditions d'optimalité

2.1 Pourquoi avons-nous besoin de conditions d'optimalité ?

Afin d'analyser ou de résoudre de manière efficace un problème d'optimisation, il est fondamental de pouvoir disposer de conditions d'optimalité. En effet, celles-ci nous servent non seulement à vérifier la validité des solutions obtenues, mais souvent l'étude de ces conditions aboutit au développement des algorithmes de résolution eux-mêmes.

Des conditions équivalentes peuvent être obtenues de diverses manières, en procédant à des analyses suivants différentes "lignes directrices". L'approche considérée ici pour l'obtention de conditions est basée sur les notions de descente et de direction admissible.

2.2 Cas sans contraintes

2.2.1 Conditions nécessaires

Etant donné un vecteur x^*, nous souhaiterions être capables de déterminer si ce vecteur est un minimum local ou global de la fonction f. La propriété de différentiabilité continue de f fournit une première manière de caractériser une solution optimale. Enonçons tout d'abord un théorème, sur lequel s'appuiera le corollaire suivant pour établir une première condition nécessaire d'optimalité :

Théorème 2.2.1. Soient $f : \mathbb{R}^n \mapsto \mathbb{R}$ continuellement différentiable et $x^* \in \mathbb{R}^n$. S'il existe un vecteur d tel que la dérivée directionnelle de f dans la direction d au point x^* (notée $f'(x^*; d)$) est strictement inférieure à zéro, alors d est une direction de descente de f en x^* (le lecteur désireux de trouver une définition de cette dernière notion est invité à se reporter à la section 3.1).

En outre, selon, une propriété connue de l'analye,

$$f'(x, d) = \nabla f(x) \cdot d.$$

Le théorème 2.2.1 peut donc être également énoncé sous la forme équivalente suivante : s'il existe un vecteur d tel que $\nabla f(x^*) \cdot d < 0$, alors d est une direction de descente de f en x^*.

Corollaire (condition nécessaire du premier ordre). Soient $f : \mathbb{R}^n \mapsto \mathbb{R}$ continuellement différentiable et $x^* \in \mathbb{R}^n$. Si x^* est un minimum local de f, alors $\nabla f(x^*) = 0$.

Ce corollaire établit une première condition nécessaire pour que x^* soit un minimum (local ou global) de la fonction f, qui est $\nabla f(x^*) = 0$. Elle fait intervenir le vecteur gradient de f, dont les composantes sont ses premières dérivées partielles ; c'est la raison pour laquelle cette condition est appelée *condition nécessaire du premier ordre.*

Remarque : si f est convexe, la condition nécessaire du premier ordre est également suffisante pour que x^* soit un minimum global.

Dans le cas où f est deux fois continuellement différentiable, une autre condition nécessaire est donnée par le théorème 2.2.2. Elle est appelée *condition nécessaire du second ordre* car elle fait intervenir la matrice hessienne de f (que nous noterons $\nabla^2 f$), dont les éléments sont ses secondes dérivées partielles.

Théorème 2.2.2 (**condition nécessaire du second ordre**)**.** Soient $f : \mathbb{R}^n \mapsto \mathbb{R}$ deux fois continuellement différentiable et $x^* \in \mathbb{R}^n$. Si x^* est un minimum local de f, alors $\nabla f(x^*) = 0$ et $\nabla^2 f(x^*)$ est semi-définie postitive.

2.2.2 Conditions suffisantes

Les conditions données précédemment sont nécessaires (si f n'est pas convexe), c'est-à-dire qu'elle doivent être satisfaites pour tout minimum local ; cependant, tout vecteur vérifiant ces conditions n'est pas nécessairement un minimum local. Le théorème 2.2.3 établit une condition suffisante pour qu'un vecteur soit un minimum local, si f est deux fois continuellement différentiable.

Théorème 2.2.3 (**condition suffisante du second ordre**)**.** Soient $f : \mathbb{R}^n \mapsto \mathbb{R}$ deux fois continuellement différentiable et $x^* \in \mathbb{R}^n$. Si $\nabla f(x^*) = 0$ et $\nabla^2 f(x^*)$ est définie positive, alors x^* est un minimum local de f.

2.3 Cas avec contraintes

Nous allons maintenant discuter des conditions d'optimalité pour le problème de minimisation de $f(x)$ sous contrainte $x \in X \subset \mathbb{R}^n$, où X est défini par une collection de m inégalités et de r égalités :

$$\begin{array}{ll} \text{minimiser } f(x) & \\ \text{sous contraintes} & g_1(x) \leq 0, ..., g_m(x) \leq 0, \\ & h_1(x) = 0, ..., h_r(x) = 0. \end{array}$$

Donnons tout d'abord la définition de la notion de direction admissible, puis nous énoncerons un théorème qui établira une première condition nécessaire (tirée d'une propriété "géométrique") d'optimalité pour un tel problème.

Définition 2.3.1 (**direction admissible**)**.** Soient un ensemble $X \subset \mathbb{R}^n$, avec $X \neq \emptyset$, et $x^* \in X$. Une *direction admissible* de X en x^* est un vecteur d tel que

$$d \neq 0 \text{ et } x^* + \alpha d \in X, \qquad \forall \alpha \in [0, \alpha_{max}] \text{ pour un certain } \alpha_{max} > 0.$$

Les conditions d'optimalité énoncées plus bas s'obtiennent suite au constat suivant : si x^* est un minimum local de f sur X, alors il ne peut y avoir aucune direction de descente dans

l'ensemble des directions admissibles de X en x^* (cela peut être démontré et fournir une première condition nécessaire d'optimalité).

La condition telle que nous venons de l'énoncer ne présente malheureusement que peu d'intérêt en vue d'un usage en pratique, car si l'ensemble des directions de descente de f en x^*, disons $D = \{d \mid \nabla f(x^*) \cdot d < 0\}$, peut être exprimé en fonction du gradient de la fonction objectif, ce n'est pas nécessairement le cas de l'ensemble de toutes les directions admissibles de X en x^* $A = \{d \mid d \neq 0 \text{ et } x^* + \alpha d \in X,\ \forall \alpha \in [0, \alpha_{max}] \text{ pour un certain } \alpha_{max} > 0\}$. Il serait avantageux de trouver un moyen de la convertir en une condition plus pratiquement utilisable mettant en jeu des équations ou des inéquations.

La stratégie suivante peut être utilisée à cette fin : des sous-ensembles de A, disons G et H, peuvent être définis en fonction respectivement des gradients des g_i actives en x^*, et des h_i. Il découle ensuite du fait que $A \cap D = \emptyset$ soit une condition nécessaire d'optimalité le fait que $G \cap H \cap D = \emptyset$ le soit également.

La condition d'optimalité de Fritz John

La condition donnée ci-dessus peut être également exprimée par les relations établies au théorème suivant, attribué à Fritz John (1948).

Théorème 2.3.1 (**condition nécessaire de Fritz John**). Soient un ensemble $X \subset \mathbb{R}^n$ avec $X \neq \emptyset$, une fonction $f : \mathbb{R}^n \mapsto \mathbb{R}$ et un vecteur $x^* \in X$. Nous admettons que X est défini par une collection de m inégalités $g_i(x) \leq 0$ et de r égalités $h_i(x) = 0$ avec $g_i : \mathbb{R}^n \mapsto \mathbb{R},\ \forall i \in \{1, 2, ..., m\}$ et $h_i : \mathbb{R}^n \mapsto \mathbb{R},\ \forall i \in \{1, 2, ..., r\}$. Supposons, de plus, que f, les g_i et les h_i sont continuellement différentiables pour tout i. Soit I l'ensemble des contraintes sous forme d'inégalités actives en $x^* : I = \{\ i \mid g_i(x^*) = 0\}$. Si x^* est un minimum local, alors il existe des scalaires $\mu_0, \mu_1, ..., \mu_m$ et $\lambda_1, ..., \lambda_r$ non tous égaux à zéro tels que :

$$\mu_0 \nabla f(x^*) + \sum_{i \in I} \mu_i \nabla g_i(x^*) + \sum_{i=1}^{r} \lambda_i \nabla h_i(x^*) = 0$$
$$\mu_0, \mu_i \geq 0, \qquad \forall i \in I.$$

Remarque : les scalaires μ_i et λ_i sont appelés *multiplicateurs de Lagrange* et la fonction définie ci-dessus est parfois appelée *fonction de Lagrange*.

La condition d'optimalité de Kuhn-Tucker

Un inconvénient posé par la condition de Fritz John est le suivant : si μ_0 est égal à zéro, alors la condition ne comporte aucune information relative au gradient de la fonction objectif, ce qui signifie qu'elle ne fournit pas d'information relative à notre problème. Dans un tel cas, elle assure simplement qu'il existe une combinaison linéaire non triviale (et non négative si les contraintes sont toutes sous forme d'inégalités) des gradients des contraintes actives qui est égale au vecteur nul ; dans ce cas, elle ne nous est pas utile en pratique pour déterminer si un vecteur est optimal. Le cas où μ_0 est strictement supérieur à zéro est donc plus intéressant. Dans cette optique, des conditions ont été développées indépendamment par Kuhn et Tucker (en 1951), qui sont précisément les conditions de Fritz John, à ceci près qu'une hypothèse y a été ajoutée impliquant que μ_0 ne puisse pas être égal à zéro.

Différentes hypothèses peuvent être posées sur les contraintes afin de garantir que $\mu_0 > 0$ (de telles hypothèses sont appelées *qualification des contraintes*). Dans le théorème énoncé ci-dessous, l'on impose que les gradients des contraintes sous formes d'égalités et des contraintes sous forme

d'inégalités actives au point considéré soient linéairement indépendants, ce qui signifie qu'il ne peut exister de combinaison linéaire non triviale de ceux-ci dont la somme vaut le vecteur nul. Sous cette hypothèse supplémentaire, la condition de Fritz John ne peut être satisfaite qu'avec $\mu_0 \neq 0$. Nous pouvons donc arbitrairement choisir $\mu_0 = 1$ et énoncer le théorème suivant :

Théorème 2.3.2 (**condition nécessaire de Kuhn-Tucker**)**.** Soient un ensemble $X \subset \mathbb{R}^n$ avec $X \neq \emptyset$, une fonction $f : \mathbb{R}^n \mapsto \mathbb{R}$ et un vecteur $x^* \in X$. Nous admettons que X est défini par une collection de m inégalités $g_i(x) \leq 0$ et de r égalités $h_i(x) = 0$ avec $g_i : \mathbb{R}^n \mapsto \mathbb{R},\ \forall i \in \{1, 2, ..., m\}$ et $h_i : \mathbb{R}^n \mapsto \mathbb{R},\ \forall i \in \{1, 2, ..., r\}$. Soit I l'ensemble des contraintes sous forme d'inégalités actives en $x^* : I = \{\ i \mid g_i(x^*) = 0\}$. Supposons, de plus, que f, les g_i et les h_i sont continuellement différentiables pour tout i et que $\nabla g_i(x^*)$ pour $i \in I$ et $\nabla h_i(x^*)$ pour $i = 1, ..., r$ sont linéairement indépendants. Si x^* est un minimum local, alors il existe des scalaires $\mu_1, ..., \mu_m$ et $\lambda_1, ..., \lambda_r$ tels que :

$$\nabla f(x^*) + \sum_{i \in I} \mu_i \nabla g_i(x^*) + \sum_{i=1}^{r} \lambda_i \nabla h_i(x^*) = 0$$
$$\mu_i \geq 0, \qquad \forall i \in I.$$

Remarque : la condition nécessaire de Kuhn-Tucker est également suffisante si f et les g_i sont convexes.

Chapitre 3

Méthodes itératives d'optimisation sans contraintes

Au travers du présent chapitre et du suivant, on s'attachera dorénavant à la description plus spécifique des algorithmes itératifs (ou méthodes itératives) qui ont été implémentés et qui permettent la résolution des problèmes d'optimisation non linéaire.

Il convient de souligner que la plupart des algorithmes d'optimisation, contrainte ou non, fonctionnent selon un schéma général consistant, à chaque itération, à se rapprocher du minimum par la résolution d'un sous-problème de minimisation. Evidemment, cette stratégie n'a de sens que si ces sous-problèmes sont plus faciles à résoudre que le problème original. Nous verrons que c'est le cas, ces derniers se révélant être notamment des problèmes de minimisation non linéaires à une seule dimension, ou encore des programmes linéaire (ceci plus particulièrement dans les algorithmes traités au chapitre suivant).

Nous considérons ici les méthodes permettant de résoudre un problème d'optimisation sans contraintes (appelées aussi parfois *méthodes d'optimisation directe*), soit le problème

$$\begin{aligned} &\text{minimiser } f(x) \\ &\text{avec } x \in \mathbb{R}^n \end{aligned}$$

pour lequel nous commencerons par décrire les méthodes suivantes :

1. Les méthodes basées sur le gradient
2. Les méthodes de Newton
3. Les méthodes utilisant des directions conjuguées
4. La méthode du simplexe (ou méthode de Nelder et Mead)

Ces méthodes utilisent des dérivées (et donc la propriété de différentiabilité de f) à l'exception des méthodes de directions conjuguées (sauf dans le cas particulier de la méthode du gradient conjugué) et de la méthode du simplexe basée, elle, sur des propriétés plus géométriques.

Après les avoir décrites, nous discuterons plus précisément des champs d'application de chacune de ces méthodes et les analyserons du point de vue de leur convergence (nous donnerons notamment, pour chacune des méthodes les conditions sous lesquelles elles convergent, ou encore déterminerons si l'optimum est atteint en un nombre fini d'itérations).

Après avoir utilisé les méthodes en pratique grâce au logiciel développé au cours de ce projet, nous serons en mesure d'établir une étude comparative des différentes méthodes, du point de vue de la performance et de la rapidité de convergence des algorithmes, et cela en fonction du type de problème et du point initial.

3.1 La méthode de descente basée sur le gradient

3.1.1 Description de la méthode

Les méthodes basées sur le gradient de la fonction objectif sont des procédures parmi les plus fondamentales pour minimiser une fonction différentiable de $\mathbb{R}^n$ dans $\mathbb{R}$. Comme la plupart des autres méthodes développées pour ce problème, elles reposent sur la propriété dite de *descente itérative*. Rappelons qu'un algorithme itératif part d'un vecteur $x^0 \in \mathbb{R}^n$ et génère une suite de vecteur $x^1, x^2, ...$ de $\mathbb{R}^n$, la propriété de descente itérative impliquant que le coût des vecteurs ainsi générés décroisse à chaque itération :

$$f(x^{k+1}) < f(x^k) \qquad \forall k \in \mathbb{N}.$$

Les méthodes basées sur le gradient appartiennent à une importante classe d'algorithmes, où les vecteurs sont générés de la manière suivante :

$$x^{k+1} = x^k + \alpha^k d^k.$$

Il paraît approprié de donner ici la définition de la notion de direction de descente :

Définition 3.1.1 (direction de descente). Soient $f : \mathbb{R}^n \mapsto \mathbb{R}$ et $x \in \mathbb{R}^n$. Un vecteur d est appelé une *direction de descente* de f en x si $\exists \alpha_{max} > 0$ tel que $f(x+\alpha d) < f(x)\ \forall \alpha \in [0, \alpha_{max}]$.

Ainsi, afin d'assurer que la propriété de descente itérative soit garantie par l'algorithme, la direction d^k choisie, dans l'équation ci-dessus, doit être une direction de descente. Selon le théorème 2.1.1, toute direction d^k telle que $\nabla f(x^k) \cdot d^k < 0$ est une direction de descente de f en x^k. Les algorithmes de ce type sont appelés *méthodes du gradient* en raison de cette relation entre la direction d^k et le gradient $\nabla f(x^k)$.

Il n'y a pas de nom universellement reconnu et utilisé pour ces méthodes. Certains auteurs réservent l'appellation "méthode du gradient" au cas particulier où $d^k = -\nabla f(x^k)$ (celle-ci est également parfois appelée *méthode optimale du gradient*), alors que d'autre l'emploient afin de désigner l'ensemble des algorithmes de ce type, pour lesquels de nombreuses stratégies de choix de d^k de sont possibles.

Venons-en maintenant à la méthode du gradient telle qu'elle a été implémentée au cours de projet : il s'agit précisément du cas particulier ou $d^k = -\nabla f(x^k)$; c'est pourquoi dorénavant, dans le présent document, l'appellation "méthode du gradient" sera utilisée uniquement en référence à ce cas précis où d^k est choisie ainsi et où α^k est déterminé suivant la politique dite de minimisation (voir ci-dessous). C'est une propriété intéressante de la direction $-\nabla f(x^k)$ qui nous conduit à ce choix : parmi toutes les direction $d \in \mathbb{R}^n$ normalisées, il s'agit de celle qui minimise la dérivée directionnelle $\nabla f(x^k) \cdot d$ de f en x^k, comme nous allons le voir dans la proposition ci-après. Ainsi, la direction que nous choisissons parmi toutes les directions $d \in \mathbb{R}^n$ telles que $|d| = 1$ est celle qui minimise la variation de f dans la direction d lorsque α tend vers zéro.

Le problème de recherche de cette direction consiste à trouver la direction d qui minimise $\nabla f(x) \cdot d$ sous contrainte $|d| = 1$. La proposition ci-dessous stipule que la direction $d = -\nabla f(x)/|\nabla f(x)|$ est la solution optimale de ce problème.

Proposition 3.1.1. Soient $f : \mathbb{R}^n \mapsto \mathbb{R}$ continuellement différentiable et $x \in \mathbb{R}^n$. Supposons que $\nabla f(x) \neq 0$. Alors le problème consistant à minimiser $f'(x, d)$ sous contrainte $|d| = 1$ a pour solution optimale $d^* = -\nabla f(x)/|\nabla f(x)|$.

La direction de descente choisie sera donc, à chaque itération, $d^k = -\nabla f(x^k)/|\nabla f(x^k)|$. Les points sont ainsi successivement générés par la méthode du gradient de la manière suivante :

$$x^{k+1} = x^k - \alpha^k \nabla f(x^k), \qquad \alpha^k > 0.$$

Le point x^0 peut être choisi arbitrairement. Il importe désormais de choisir α^k d'une manière aussi convenable que possible. Tout comme pour d^k, diverses stratégies peuvent être employées pour ce choix. Celle que nous adoptons ici est généralement désignée sous l'appellation de *règle de minimisation*. Elle consiste à choisir, à chaque itération, α^k comme étant la solution optimale du problème de minimisation monodimentionnelle de f le long de la demi-droite définie par le point x^k et la direction $-\nabla f(x^k)$. Un tel sous-problème peut être résolu de diverses manières, par exemple par la méthode de Newton.

Remarquons que la méthode s'arrête lorsque $\nabla f(x^k) = 0$ car dans ce cas $x^{k+1} = x^k$. Nous allons aborder dans la section suivante les questions plus particulièrement relatives à la convergence de cette méthode.

3.1.2 Convergence de la méthode

Idéalement, nous souhaiterions pouvoir générer, à l'aide de la méthode du gradient, une séquence $\{x^k\}$ convergeant vers un minimum global de f. Cependant, c'est bien sûr trop demander à une telle méthode, du moins si f n'est pas convexe (en raison de la présence d'extrêma locaux qui ne sont pas globaux). La méthode du gradient est guidée localement selon la forme de f dans la région correspondante au point x^k, et peut ainsi être attirée par tout type de minimum, qu'il soit local ou global. Remarquons que si, pour une quelconque raison, la méthode est démarrée depuis ou rejoint un maximum ou un point stationnaire, elle se termine en ce point. Ainsi, si f n'est pas convexe, nous pouvons, au mieux, attendre de la méthode du gradient qu'elle converge vers un point stationnaire.

Le résultat suivant peut ainsi être démontré :

Théorème 3.1.1 (**convergence de la méthode du gradient**)**.** Soit $\{x^k\}$ une séquence générée par la méthode du gradient. Alors tout point limite de $\{x^k\}$ est un point stationnaire.

Il peut arriver que la méthode du gradient converge de manière finie, mais ce n'est en général pas le cas. Il est donc nécessaire d'utiliser un critère permettant d'arrêter l'exécution lorsque x^k est suffisamment proche d'un point stationnaire, par exemple $|\nabla f(x^k)| < \epsilon$, où ϵ est un scalaire positif arbitrairement choisi. A priori, la valeur que nous devons fixer pour ϵ dépend du problème considéré.

L'inconvénient majeur de la méthode du gradient survient lorsque les surfaces de coût égal de f sont "allongées", et que x^k est tel que la direction du gradient y est presque orthogonale à la direction menant au minimum. Celle-ci adopte alors le comportement bien connu du "zig-zag" et progresse extrêmement lentement, comme nous aurons l'occasion de le constater lors de l'application pratique de la méthode.

Il existe des moyens de surmonter ces difficultés en choisissant la direction d^k un peu différemment. La direction donnée par le gradient peut être corrigée en la multipliant par une matrice ou en y ajoutant un vecteur approprié : la performance peut être améliorée en se déplaçant dans la direction $d^k = -D\nabla f(x^k)$ ou $d^k = \nabla f(x^k) + v$, où D et v sont convenablement choisis. Cependant, nous n'entrerons ici pas plus dans les détails, car la méthode du gradient qui a été implémentée est celle que nous avons précédemment décrite.

3.2 La méthode de Newton

Alors que la méthode du gradient utilise une approximation linéaire pour trouver une direction de mouvement, l'idée de la méthode itérative de Newton est de minimiser à chaque itération l'approximation quadratique de f au point courant x^k et donnée par le développement de Taylor d'ordre 2 :

$$q^k(x) = f(x^k) + \nabla f(x^k) \cdot (x - x^k) + \frac{1}{2}(x - x^k) \cdot \nabla^2 f(x^k)(x - x^k).$$

Une condition nécessaire pour que le minimum de $q^k(x)$ soit atteint est $\nabla q^k(x) = 0$ soit

$$\nabla f(x^k) + \nabla^2 f(x^k)(x - x^k) = 0.$$

Le vecteur généré à l'itération $k + 1$ est le vecteur minimisant $q^k(x)$, c'est-à-dire le vecteur satisfaisant l'équation précédente, soit

$$x^{k+1} = x^k - (\nabla^2 f(x^k))^{-1} \nabla f(x^k).$$

La méthode nécessitant l'évalutation de la matrice hessienne de f, elle ne peut être utilisée que si f est deux fois continuellement différentiable (le méthode de Newton requiert même l'évaluation de l'inverse de cette matrice, ce qui est coûteux en terme de calculs). On peut remarquer que la méthode s'arrête également lorsque $\nabla f(x^k) = 0$, car il s'ensuit que $x^{k+1} = x^k$. Si, en plus, $\nabla^2 f(x^*)$ est définie positive, alors la condition suffisante donnée au théorème 2.2.3 est satisfaite, impliquant que x^k soit un minimum local.

Notons que la méthode de Newton converge en une seule itération si f est quadratique.

3.2.1 Convergence de la méthode de Newton

La méthode de Newton décrite ci-dessus présente plusieurs inconvénients :

1. L'inverse de la matrice hessienne $(\nabla^2 f(x^k))^{-1}$ peut ne pas exister, auquel cas la méthode échoue. Cela intervient typiquement lorsque la méthode atteint une région où f est linéaire (ses secondes dérivées partielles valent zéro).
2. La méthode de Newton n'est pas une méthode de descente : il est possible que $f(x^{k+1})$ soit supérieur à $f(x^k)$.
3. Elle est attirée aussi bien par les minima que par les maxima locaux (cette propriété est liée à la précédente). En effet, la méthode, à chaque itération, recherche uniquement un point tel que le gradient de l'approximation quadratique soit égal au vecteur nul, que ce soit point soint un maximum, un minimum ou un point stationnaire.

La méthode ne converge donc pas en général, notamment si elle est démarrée loin d'un minimum local, pour les première et troisième raisons. Cependant, elle converge sous certaines restrictions : si elle est exécutée à partir d'un point suffisamment proche d'un minimum local et que $\nabla^2 f(x^k)$ n'est pas singulière, alors la méthode de Newton convergera vers ce minimum (mais pas de manière finie, de sorte qu'une condition d'arrêt est requise de façon analogue à la méthode du gradient).

3.3 Les méthodes utilisant des directions conjuguées

3.3.1 Description de la méthode

Ces méthode sont basées sur l'important concept de la conjugaison et ont été développées afin de résoudre le problème quadratique

$$\text{minimiser } f(x) = \frac{1}{2} x \cdot Qx + b \cdot x + c$$
$$\text{avec } x \in \mathbb{R}^n$$

où $Q \in \mathbb{R}^{n \times n}$ est symétrique et définie positive, $b \in \mathbb{R}^n$ et $c \in \mathbb{R}$. Les méthodes de direction conjuguées peuvent résoudre les problèmes de cette forme en au plus n itérations et, contrairement au méthodes présentées jusqu'à présent, elle n'utilisent pas de dérivées, sauf dans le cas particulier de la méthode du gradient conjugué. Donnons la définition de la notion de "conjugaison" :

Définition 3.3.1. Soient Q une matrice $n \times n$ symétrique et définie positive et un ensemble de directions non nulles $\{d^1, d^2, ..., d^k\}$. Ces directions sont dites *Q-conjuguées* si

$$d^i \cdot Qd^j = 0, \qquad \forall i, j \text{ tels que } i \neq j.$$

Propriété 3.3.1. Si $d^1, ..., d^k$ sont Q-conjuguées, alors elles sont linéairement indépendantes.

Propriété 3.3.2. Comme des direction Q-conjuguées sont linéairement indépendantes, alors l'espace vectoriel engendré par un ensemble de n directions Q-conjuguées est de dimension n.

Etant donné un ensemble de n directions Q-conjuguées $d^0, d^1, ..., d^{n-1}$, la méthode de directions conjuguées correspondante est donnée par

$$x^{k+1} = x^k + \alpha^k d^k, \qquad k = 0, ..., n-1$$

où x^0 est un vecteur de départ choisi arbitrairement et où les α^k sont obtenus par minimisation monodimentionnelle le long de d^k.

Le principal résultat concernant les méthodes utilisant des directions conjuguées est qu'à chaque itération k, la méthode minimise f sur le sous-espace généré par les k premières directions Q-conjuguées utilisées par l'algorithme. A la $n^{\text{ième}}$ itération au plus tard, ce sous-espace inclura alors le minimum global de f, grâce à la propriété d'indépendance linéaire des direction Q-conjuguées qui assure qu'à l'itération n, l'espace vectoriel généré par les n directions Q-conjuguées ne sera autre que $\mathbb{R}^n$.

Comment construire les directions Q-conjuguées ?

Des directions Q-conjuguées $d^0, ..., d^k$ peuvent être générées à partir d'un ensemble de vecteurs linéairement indépendants $\xi^0, ..., \xi^k$ en utilisant la procédure dite *de Gram-Schmidt*, de telle sorte que pour tout i entre 0 et k, le sous-espace généré par $d^0, ..., d^i$ soit égale au sous-espace généré par $\xi^0, ..., \xi^i$. Cette procédure fonctionne de la manière suivante. Elle commence par choisir $d^0 = \xi^0$. Supposons maintenant que les directions $d^0, ...d^i$ on été construites ($i < k$), satisfaisant la propriété précédente. Alors d^{i+1} est construite comme suit :

$$d^{i+1} = \xi^{i+1} + \sum_{m=0}^{i} c^{(i+1)m} d^m$$

Nous pouvons noter que si d^{i+1} est construite d'une telle manière, elle est effectivement linéairement indépendante de $d^0, ..., d^i$. En effet, le sous-espace généré par les directions $d^0, ..., d^i$ est le même que le sous-espace généré par les directions $\xi^0, ..., \xi^i$, et ξ^{i+1} est linéairement indépendante de $\xi^0, ..., \xi^i$. ξ^{i+1} ne fait donc pas partie du sous-espace généré par les combinaisons linéaires de

la forme $\sum_{m=0}^{i} c^{(i+1)m} d^m$, de sorte que d^{i+1} n'en fait pas partie non plus et est donc linéairement indépendante des $d^0, ..., d^i$.

Les coefficients $c^{(i+1)m}$, eux, sont choisis de manière à assurer la Q-conjugaison des $d^0, ..., d^{i+1}$, c'est-à-dire de telle sorte que pour tout $j = 0, ..., i$, nous ayons

$$d^{i+1} \cdot Qd^j = 0.$$

La méthode du gradient conjugué

La méthode du gradient conjugué est obtenue en appliquant la procédure de Gram-Schmidt aux gradients $\nabla f(x^0), ..., \nabla f(x^{n-1})$, c'est-à-dire en posant $\xi^0 = -\nabla f(x^0), ..., \xi^{n-1} = -\nabla f(x^{n-1})$. En outre, nous avons que

$$\nabla f(x) = Qx + b$$

et

$$\nabla^2 f(x) = Q.$$

Notons que la méthode se termine si $\nabla f(x^k) = 0$. La particularité intéressante de la méthode du gradient conjugué est que le membre de droite de l'équation donnant la valeur de d^{k+1} dans la procédure de Gram-Schmidt peut être grandement simplifié. En particulier, nous pouvons vérifier que seul un des termes de la somme est non nul, de sorte que cette méthode, implémentée, se révèle ainsi plus rapide que la méthode décrite précédemment qui applique la procédure de Gram-Schmidt à des vecteur linéairement indépendants quelconques.

3.3.2 Convergence des méthodes de directions conjuguées

Comme nous le mentionnions préalablement, le théorème suivant peut être démontré :

Théorème 3.3.1. Soit x^n le point obtenu à la $n^{\text{ième}}$ itération d'une méthode de directions conjuguées, sous les hypothèses décrites à la section 3.3.1, en partant de x^0 quelconque. Alors x^n est un minimum global de f sur $\mathbb{R}^n$.

La méthode converge donc de manière finie, pour peu que les hypothèses précédentes soient respectées.

Il convient de mentionner qu'il est également possible d'employer les méthodes de directions conjuguées pour résoudre des problèmes non quadratiques. Cependant, il faut pour cela recourir à des stratégies permettant de surmonter la "perte" de conjugaison des directions générées due à la présence de termes non quadratiques dans la fonction objectif. Toutefois, de telles implémentations n'ayant pas été réalisées dans le cadre de ce projet, la description de telles méthodes est au-delà du cadre de ce document.

3.4 La méthode du simplexe

C'est un euphémisme que de dire que la méthode du simplexe (à ne pas confondre avec la méthode du simplexe pour la programmation linéaire), également appelée méthode de Nelder et Mead, fonctionne différement des autres méthodes décrites jusqu'ici. En effet, elle ne consiste pas à déterminer, à chaque itération, une direction le long de laquelle se déplacer depuis x^k pour obtenir x^{k+1}. Son principe est de maintenir un ensemble de $n + 1$ points, appelé *simplexe*. A chaque itération, l'algorithme remplace le point de coût maximum x_{max} dans le simplexe par un autre dont le coût est inférieur. Ce nouveau point est déterminé par l'algorithme d'une manière très "géométrique", suivant les étapes de réflexion, d'expansion ou de contraction qui sont décrites ci-dessous.

Décrivons précisément l'algorithme qui a été implémenté. Avant le début de l'exécution, le choix d'un simplexe initial doit être opéré. Soient $x^0, x^1, ..., x^n$ les $n+1$ points du simplexe courant. Soient x_{min} et x_{max} les points de coût respectivement le moins et le plus élevé, c'est-à-dire tels que

$$f(x_{min}) = \min_{i=0,...,n} f(x^i)$$

et

$$f(x_{max}) = \max_{i=0,...,n} f(x^i).$$

Soit $\bar{x}$ défini de la manière suivante (nous pourrions le qualifier de "centre" du polyèdre formé par les points du simplexe à l'exception de x_{max}) :

$$\bar{x} = \frac{1}{n}(-x_{max} + \sum_{i=0}^{n} x^i).$$

L'itération consiste à remplacer le pire des points x_{max} par un point de coût inférieur ; pour cela, le point dit de réflection x_{ref} est calculé, situé sur la droite passant par x_{max} et $\bar{x}$, et symétrique à x_{max} par rapport à $\bar{x}$:

$$x_{ref} = 2\bar{x} - x_{max}.$$

Alors, un nouveau point x_{new} amené à remplacer x_{max} dans le simplexe est calculé, en fonction des coûts de x_{ref} et des points du simplexe autres que x_{max}. En fonction de cela, x_{new} pourra être déterminé selon trois phases différentes, décrites ci-dessous :

1. Si $f(x_{min}) > f(x_{ref})$, l'**expansion** est effectuée.
   Le point x_{exp} est calculé :

   $$x_{exp} = 2x_{ref} - \bar{x}$$

   et x_{new} est défini comme suit :

   $$x_{new} = \begin{cases} x_{exp} & \text{si } f(x_{exp}) < f(x_{ref}), \\ x_{ref} & \text{sinon.} \end{cases}$$

2. Si $\max_{x^i \neq x_{max}} f(x^i) > f(x_{ref}) \geq f(x_{min})$, la **réflection** est effectuée.
   x_{new} est simplement défini comme étant égal à x_{ref}.

3. Si $f(x_{ref}) \geq \max_{x^i \neq x_{max}} f(x^i)$, la **contraction** est effectuée.
   x_{new} est calculé comme suit :

   $$x_{new} = \begin{cases} \frac{1}{2}(x_{max} + \bar{x}) & \text{si } f(x_{max}) \leq f(x_{ref}), \\ \frac{1}{2}(x_{ref} + \bar{x}) & \text{sinon.} \end{cases}$$

Dans les trois cas, le nouveau simplexe est formé en remplaçant x_{max} par x_{new}.
Il n'existe pas de résultat connu quant à la convergence de cette méthode.

3.5 Etude comparative des méthodes d'optimisation sans contraintes

Dans cette section, nous allons tâcher de résoudre des problèmes de manière réelle grâce à l'application développée, et comparer les résultats obtenus pour les différentes méthodes.

Soit le problème quadratique P_1 :

$$\text{minimiser } f(x) = 4x_1^2 + 4x_2^2 - 12x_2 - 4x_1x_2$$

On résout ce problème par la méthode du gradient. Le point de départ est $x_1(-20, 15)$ et la condition d'arrêt $|\nabla f(x^k)| < 0.01$. Le résultat est donné dans le tableau 3.1 (les valeurs ont été arrondies à deux décimales pour une lecture plus confortable) :

k	x^k	$f(x^k)$	$\nabla f(x^k)$	$\|\nabla f(x^k)\|$	$d^k = -\nabla f(x^k)$	α^k	x^{k+1}
1	(-20.0, 15.0)	3520	(-220.0, 188.0)	289.39	(220.0, -188.0)	0.08	(-1.59, -0.73)
2	(-1.59, -0.73)	16.39	(-9.81, -11.48)	15.1	(9.81, 11.48)	0.25	(0.83, 2.1)
3	(0.83, 2.1)	-11.77	(-1.77, 1.51)	2.33	(1.77, -1.51)	0.08	(0.98, 1.98)
4	(0.98, 1.98)	-12	(-0.08, -0.09)	0.12	(0.08, 0.09)	0.25	(1.0, 2.0)
5	(1.0, 2.0)	-12	(-0.01, 0.01)	0.02	(0.01, -0.01)	0.08	(1.0, 2.0)
6	(1.0, 2.0)	-12	(0.0, 0.0)	0	-	-	-

TAB. 3.1 – *exécution de la méthode du gradient pour le problème P_1 depuis (-20,15).*

Un premier constat est que la méthode du gradient n'a pas eu besoin de beaucoup d'itérations pour trouver la solution optimale de ce problème en partant de ce point initial. Cela est dû en partie au fait que celui-ci se trouve judicieusement placé : les lignes de coût égal de f sont des ellipses centrées autour du minimum et, à chaque itération, x^k se trouve proche d'un point d'intersection de l'ellipse avec l'un de ses axes ; il s'ensuit que, depuis de tels points, la direction menant au minimum est proche de la direction donnée par le gradient, permettant à la méthode de progresser rapidement. Si nous résolvons le même problème en partant du point $(2, 27)$, nous obtenons :

k	x^k	$f(x^k)$	$\nabla f(x^k)$	$\|\nabla f(x^k)\|$	$d^k = -\nabla f(x^k)$	α^k	x^{k+1}
1	(2.0, 27.0)	2392	(-92.0, 196.0)	216.52	(92.0, -196.0)	0.09	(10.31, 9.31)
2	(10.31, 9.31)	275.93	(45.22, 21.23)	49.95	(-45.22, -21.23)	0.2	(1.12, 4.99)
3	(1.12, 4.99)	22.49	(-11.02, 23.48)	25.93	(11.02, -23.48)	0.09	(2.11, 2.88)
4	(2.11, 2.88)	-7.87	(5.42, 2.54)	5.98	(-5.42, -2.54)	0.2	(1.01, 2.36)
5	(1.01, 2.36)	-11.51	(-1.32, 2.81)	3.11	(1.32, -2.81)	0.09	(1.13, 2.1)
6	(1.13, 2.1)	-11.94	(0.65, 0.3)	0.72	(-0.65, -0.3)	0.2	(1.0, 2.04)
7	(1.0, 2.04)	-11.99	(-0.16, 0.34)	0.37	(0.16, -0.34)	0.09	(1.02, 2.01)
8	(1.02, 2.01)	-12	(0.08, 0.04)	0.09	(-0.08, -0.04)	0.2	(1.0, 2.01)
9	(1.0, 2.01)	-12	(-0.02, 0.04)	0.04	(0.02, -0.04)	0.09	(1.0, 2.0)
10	(1.0, 2.0)	-12	(0.01, 0.0)	0.01	(-0.01, 0.0)	0.2	(1.0, 2.0)
11	(1.0, 2.0)	-12	(0.0, 0.0)	0.01	-	-	-

TAB. 3.2 – *exécution de la méthode du gradient pour le problème P_1 depuis (2,27).*

La méthode a eu besoin de presque deux fois plus d'itérations que lors de la première exécu-

tion, soit 5 de plus, pour trouver une approximation du minimum $(1, 2)$ d'une précision équivalente alors que la distance entre le point initial et le minimum était sensiblement la même. Nous l'expliquons par le fait que les x^k ne sont plus aussi proches des intersections de l'ellipse avec ses axes, ce qui implique qu'à chaque itération, la direction du gradient est relativement éloignée de celle menant au minimum, conduisant à une convergence plus lente.

Poussé à l'extrême, ce phénomène conduit à des exécutions au cours desquelles la direction du gradient est pratiquement orthogonale à celle menant au minimum, produisant le fameux phénomène de "zigzag", et conduisant à une convergence de la méthode considérablement lente. Cela peut être typiquement observé lorsque les courbes de niveau sont "allongées" ; la méthode du gradient se révèle alors particulièrement peu efficace.

Nous pouvons également observer la forte dépendance du comportement de la méthode du gradient à son point de départ. En effet, une conséquence du raisonnement précédent est que, pour un problème donné, il est possible qu'elle converge vite depuis un point initial et extrêmement lentement depuis un autre point.

Nous exécutons maintenant les algorithmes de Newton, des directions conjuguée, du gradient conjugué et du simplexe pour résoudre ce problème. Le résultats obtenu pour la méthode de Newton est (la condition d'arrêt est la même que précédemment, c'est-à-dire $|\nabla f(x^k)| < 0.01$) :

k	x^k	$f(x^k)$	$\nabla f(x^k)$	$\nabla^2 f(x^k)$	$(\nabla^2 f(x^k))^{-1}$	d^k	x^{k+1}
1	(-20.0, 15.0)	3520	(-220.0, 188.0)	[(8.0, -4.0), (-4.0, 8.0)]	[(0.17, 0.08), (0.08, 0.17)]	(21.0, -13.0)	(1.0, 2.0)
2	(1.0, 2.0)	-12	(0.0, 0.0)	[(8.0, -4.0), (-4.0, 8.0)]	-	-	-

TAB. 3.3 – *exécution de la méthode de Newton pour P_1 depuis (-20,15).*

où les matrices hessienne et son inverse sont données ligne par ligne. La méthode converge de manière finie en une seule itération, comme nous pouvions nous y attendre (et ceci indépendamment du point initial).

L'exécution des méthodes de directions conjuguées et du gradient conjugué donnent respectivement les résultats suivants :

k	x^k	$f(x^k)$	Vector used by G.-S.	Q-conjugate direction	α^k	x^{k+1}
1	(-20.0, 15.0)	3520	(1.0, 0.0)	(1.0, 0.0)	27.5	(7.5, 15.0)
2	(7.5, 15.0)	495	(0.0, 1.0)	(0.5, 1.0)	-13	(1.0, 2.0)

TAB. 3.4 – *exécution de la méthode des directions conjuguées pour P_1 depuis (-20,15).*

k	x^k	$f(x^k)$	Gradient vector used by G.-S.	Q-conjugate direction	α^k	x^{k+1}
1	(-20.0, 15.0)	3520	(220.0, -188.0)	(220.0, -188.0)	0.08	(-1.59, -0.73)
2	(-1.59, -0.73)	16.39	(9.81, 11.48)	(10.41, 10.97)	0.25	(1.0, 2.0)

TAB. 3.5 – *exécution de la méthode du gradient conjugué pour P_1 depuis (-20,15).*

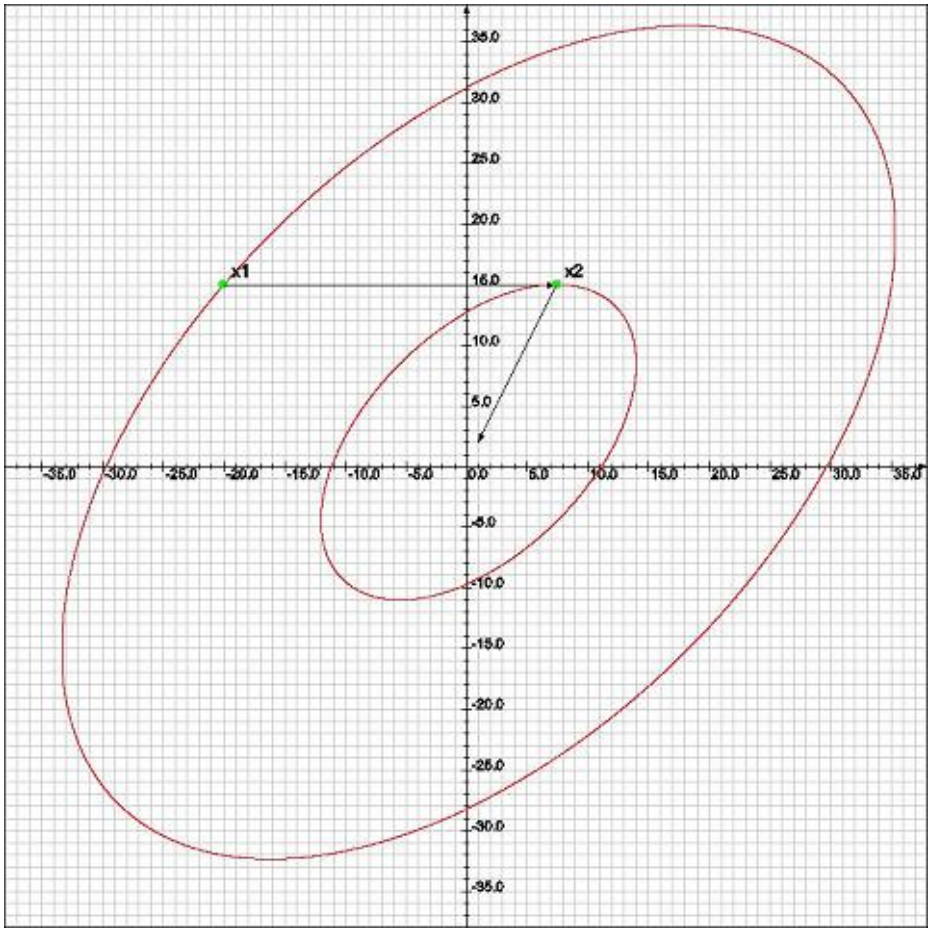

FIG. 3.1 – *illustration de la convergence finie en deux itération de la méthode des directions conjuguées pour le problème P1 en partant depuis (-20,15).*

où, lors de la première de ces deux exécution, les directions Q-conjuguées ont été générées en appliquant la procédure de Gram-Schmidt aux vecteurs linéairement indépendants (1,0) et (0,1).

On peut noter que ces méthodes ont trouvé le minimum en deux itérations, le problème étant bidimentionnel. Il n'y a pas de différence particulière à souligner entre les résultats donnés par ces deux méthodes, si ce n'est que, pour les raisons expliquées à la section 3.3, la méthode du gradient conjugué se révèle plus rapide et donc plus utile en pratique. On constatera tout de même qu'à la première itération, elles minimisent f le long de la première des deux directions linéairement indépendantes ; puis, lors de la deuxième, le minimum sur tout $\mathbb{R}^2$ est obtenu. La figure 3.1 illustre l'exécution de la méthode des directions conjuguées.

L'exécution de l'algorithme de Nelder et Mead pour ce problème est donnée dans le tableu 3.6 (la condition d'arrêt est toujours la même que pour les méthodes du gradient et de Newton), où x^k est le nouveau point déterminé à chaque itération par l'algorithme et destiné à remplacer le point de coût maximal dans le simplexe courant.

Ce dernier algorithme nécessite un nombre d'itérations plus important que les précédents. A première vue, nous pourrions déduire qu'il est moins efficace ou moins adapté à ce problème, néanmoins, nous devons garder à l'esprit qu'il ne requiert nullement la résolution d'un sous-

k	Simplexe courant	x^k	$f(x^k)$	Opération réalisée
1	[(-20.0, 15.0), (-10.0, 8.0), (0.0, 10.0)]	(10.0, 3.0)	280	reflexion
2	[(10.0, 3.0), (-10.0, 8.0), (0.0, 10.0)]	(-2.5, 7.25)	220.75	contraction
3	[(10.0, 3.0), (-2.5, 7.25), (0.0, 10.0)]	(4.38, 5.81)	40.23	contraction
4	[(4.38, 5.81), (-2.5, 7.25), (0.0, 10.0)]	(1.88, 3.06)	-8.14	reflexion
5	[(4.38, 5.81), (-2.5, 7.25), (1.88, 3.06)]	(0.31, 5.84)	59.56	contraction
6	[(4.38, 5.81), (0.31, 5.84), (1.88, 3.06)]	(1.72, 5.14)	20.49	contraction
7	[(4.38, 5.81), (1.72, 5.14), (1.88, 3.06)]	(-0.78, 2.39)	4.08	reflexion
8	[(-0.78, 2.39), (1.72, 5.14), (1.88, 3.06)]	(-0.62, 0.31)	-1.02	reflexion
9	[(-0.78, 2.39), (-0.62, 0.31), (1.88, 3.06)]	(1.33, 1.34)	-8.93	contraction
10	[(1.33, 1.34), (-0.62, 0.31), (1.88, 3.06)]	(0.49, 1.26)	-10.26	contraction
11	[(1.33, 1.34), (0.49, 1.26), (1.88, 3.06)]	(1.39, 2.18)	-11.54	contraction
12	[(1.33, 1.34), (0.49, 1.26), (1.39, 2.18)]	(0.55, 2.1)	-10.98	reflexion
13	[(0.55, 2.1), (0.49, 1.26), (1.39, 2.18)]	(0.73, 1.7)	-11.67	contraction
14	[(0.55, 2.1), (0.73, 1.7), (1.39, 2.18)]	(0.81, 2.02)	-11.83	contraction
15	[(0.81, 2.02), (0.73, 1.7), (1.39, 2.18)]	(1.08, 2.02)	-11.98	contraction
16	[(0.81, 2.02), (0.73, 1.7), (1.08, 2.02)]	(0.84, 1.86)	-11.91	contraction
17	[(0.81, 2.02), (0.84, 1.86), (1.08, 2.02)]	(0.88, 1.98)	-11.95	contraction
18	[(0.88, 1.98), (0.84, 1.86), (1.08, 2.02)]	(1.05, 2.07)	-11.98	contraction
19	[(0.88, 1.98), (1.05, 2.07), (1.08, 2.02)]	(0.97, 2.01)	-12	contraction
20	[(0.97, 2.01), (1.05, 2.07), (1.08, 2.02)]	(1.05, 2.03)	-11.99	contraction
21	[(0.97, 2.01), (1.05, 2.07), (1.05, 2.03)]	(0.97, 1.97)	-12	reflexion
22	[(0.97, 2.01), (0.97, 1.97), (1.05, 2.03)]	(1.01, 2.01)	-12	contraction
23	[(0.97, 2.01), (0.97, 1.97), (1.01, 2.01)]	(1.0, 1.98)	-12	contraction
24	[(1.0, 1.98), (0.97, 1.97), (1.01, 2.01)]	(0.99, 1.98)	-12	contraction
25	[(1.0, 1.98), (0.99, 1.98), (1.01, 2.01)]	(1.0, 2.01)	-12	reflexion
26	[(1.0, 2.01), (0.99, 1.98), (1.01, 2.01)]	(0.99, 2.0)	-12	contraction
27	[(1.0, 2.01), (0.99, 2.0), (1.01, 2.01)]	(1.0, 1.99)	-12	reflexion
28	[(1.0, 1.99), (0.99, 2.0), (1.01, 2.01)]	(1.0, 2.0)	-12	contraction
29	[(1.0, 1.99), (0.99, 2.0), (1.0, 2.0)]	(1.0, 2.0)	-12	contraction
30	[(1.0, 2.0), (0.99, 2.0), (1.0, 2.0)]	(1.0, 2.0)	-12	contraction

TAB. 3.6 – *exécution de la méthode du simplexe pour* P_1.

problème de minimisation monodimentionnelle, comme c'est le cas de la méthode du gradient ni l'évaluation de l'inverse d'une matrice comme l'exige la méthode de Newton ; seules quelques opérations arithmétiques élémentaires et des comparaisons de valeurs sont nécessaires à chaque itération.

En conséquence, et après avoir résolu d'autres problèmes quadratiques, nous en concluons que chacun des algorithmes décrits précédemment peut être utilisé pour résoudre ce type de problème ; cependant, la méthode de Newton et les méthodes de directions conjuguées restent les mieux adaptées, de part leur convergence finie en un nombre fixe d'itérations. La méthode du gradient se révèle dans ce cas la moins utile, car elle converge trop lentement en certaines circonstances.

Distinguons plus précisément les différents cas rencontrés lorsque f est quadratique (notons-la $f(x) = \frac{1}{2}x \cdot Qx + b \cdot x + c$) : il est possible de montrer que si la matrice hessienne $\nabla^2 f(x) = Q$ n'est pas semi-définie positive, f ne peut avoir aucun minimum local. Si $\nabla^2 f(x)$ est semi-définie positive, f est convexe (d'après une propriété des fonctions convexes) et ainsi tout vecteur satisfaisant la condition $\nabla f(x) = Qx + b = 0$ est un minimum global de f ; si $\nabla^2 f(x)$ est semi-définie positive et inversible, l'équation $\nabla f(x) = 0$ admet une solution qui peut être déterminée par la méthode de Newton ; si $\nabla^2 f(x)$ est semi-définie positive et singulière, nous pouvons voir que

l'équation $\nabla f(x) = 0$ n'admet pas de solution et la méthode de Newton échoue. Si $\nabla^2 f(x)$ est définie positive, alors elle est inversible (d'après une propriété des matrices symétriques et définies positives), donc l'équation $\nabla f(x) = 0$ admet une solution ; de plus, f est strictement convexe, ce qui implique qu'il existe au plus un unique minimum global de f. Il s'ensuit que l'équation $\nabla f(x) = 0$ admet une solution unique qui peut être trouvée par la méthode de Newton ou les méthodes de directions conjuguées.

Soit $P2$:

$$\text{minimiser } f(x) = (x_1 - 2)^4 + (x_1 - 2x_2)^2$$

En partant d'un point relativement proche de l'optimum $x_1(0, 4)$, on résout ce problème par les méthodes du gradient et de Newton, avec la condition d'arrêt $|\nabla f(x^k)| < 0.01$ (notons qu'il est toujours possible d'affiner notre approximation du minimum en spécifant une valeur plus petite). Le résultat de l'exécution de la méthode du gradient est (nous n'afficherons pas l'exécution en entier pour des raisons de place) donné dans le tableau 3.7, et celui de la méthode de Newton dans le tableau 3.8.

k	x^k	$f(x^k)$	$\nabla f(x^k)$	$\lvert\nabla f(x^k)\rvert$	$d^k = -\nabla f(x^k)$	α^k	x^{k+1}
1	(0.0, 4.0)	80	(-48.0, 32.0)	57.69	(48.0, -32.0)	0.06	(3.03, 1.98)
2	(3.03, 1.98)	1.99	(2.49, 3.73)	4.49	(-2.49, -3.73)	0.21	(2.5, 1.19)
3	(2.5, 1.19)	0.08	(0.75, -0.5)	0.9	(-0.75, 0.5)	0.11	(2.42, 1.24)
4	(2.42, 1.24)	0.03	(0.17, 0.25)	0.3	(-0.17, -0.25)	0.33	(2.36, 1.16)
5	(2.36, 1.16)	0.02	(0.29, -0.19)	0.35	(-0.29, 0.19)	0.12	(2.33, 1.18)
6	(2.33, 1.18)	0.01	(0.08, 0.12)	0.15	(-0.08, -0.12)	0.35	(2.3, 1.14)
7	(2.3, 1.14)	0.01	(0.16, -0.11)	0.2	(-0.16, 0.11)	0.12	(2.28, 1.15)
8	(2.28, 1.15)	0.01	(0.05, 0.08)	0.09	(-0.05, -0.08)	0.37	(2.26, 1.12)
...							
32	(2.14, 1.07)	0	(0.01, 0.01)	0.01	(-0.01, -0.01)	0.4	(2.13, 1.07)
33	(2.13, 1.07)	0	(0.01, -0.01)	0.02	(-0.01, 0.01)	0.13	(2.13, 1.07)
34	(2.13, 1.07)	0	(0.01, 0.01)	0.01	-	-	-

TAB. 3.7 – *exécution de la méthode du gradient pour P_2.*

k	x^k	$f(x^k)$	$\nabla f(x^k)$	$\nabla^2 f(x^k)$	$(\nabla^2 f(x^k))^{-1}$	d^k	x^{k+1}
1	(0.0, 4.0)	80	(-48.0, 32.0)	[(50.0, -4.0), (-4.0, 8.0)]	[(0.02, 0.01), (0.01, 0.13)]	(0.67, -3.67)	(0.67, 0.33)
2	(0.67, 0.33)	3.16	(-9.48, 0.0)	[(23.33, -4.0), (-4.0, 8.0)]	[(0.05, 0.02), (0.02, 0.14)]	(0.44, 0.22)	(1.11, 0.56)
3	(1.11, 0.56)	0.62	(-2.81, 0.0)	[(11.48, -4.0), (-4.0, 8.0)]	[(0.11, 0.05), (0.05, 0.15)]	(0.3, 0.15)	(1.41, 0.7)
4	(1.41, 0.7)	0.12	(-0.83, 0.0)	[(6.21, -4.0), (-4.0, 8.0)]	[(0.24, 0.12), (0.12, 0.18)]	(0.2, 0.1)	(1.6, 0.8)
5	(1.6, 0.8)	0.02	(-0.25, 0.0)	[(3.87, -4.0), (-4.0, 8.0)]	[(0.53, 0.27), (0.27, 0.26)]	(0.13, 0.07)	(1.74, 0.87)
6	(1.74, 0.87)	0	(-0.07, 0.0)	[(2.83, -4.0), (-4.0, 8.0)]	[(1.2, 0.6), (0.6, 0.43)]	(0.09, 0.04)	(1.82, 0.91)
7	(1.82, 0.91)	0	(-0.02, 0.0)	[(2.37, -4.0), (-4.0, 8.0)]	[(2.7, 1.35), (1.35, 0.8)]	(0.06, 0.03)	(1.88, 0.94)
8	(1.88, 0.94)	0	(-0.01, 0.0)	[(2.16, -4.0), (-4.0, 8.0)]	-	-	-

TAB. 3.8 – *exécution de la méthode de Newton pour P_2.*

La méthode du gradient a eu besoin de 34 itérations pour trouver une approximation du minimum, alors que la méthode de Newton n'en a nécessité que 8. Autre différence remarquable :

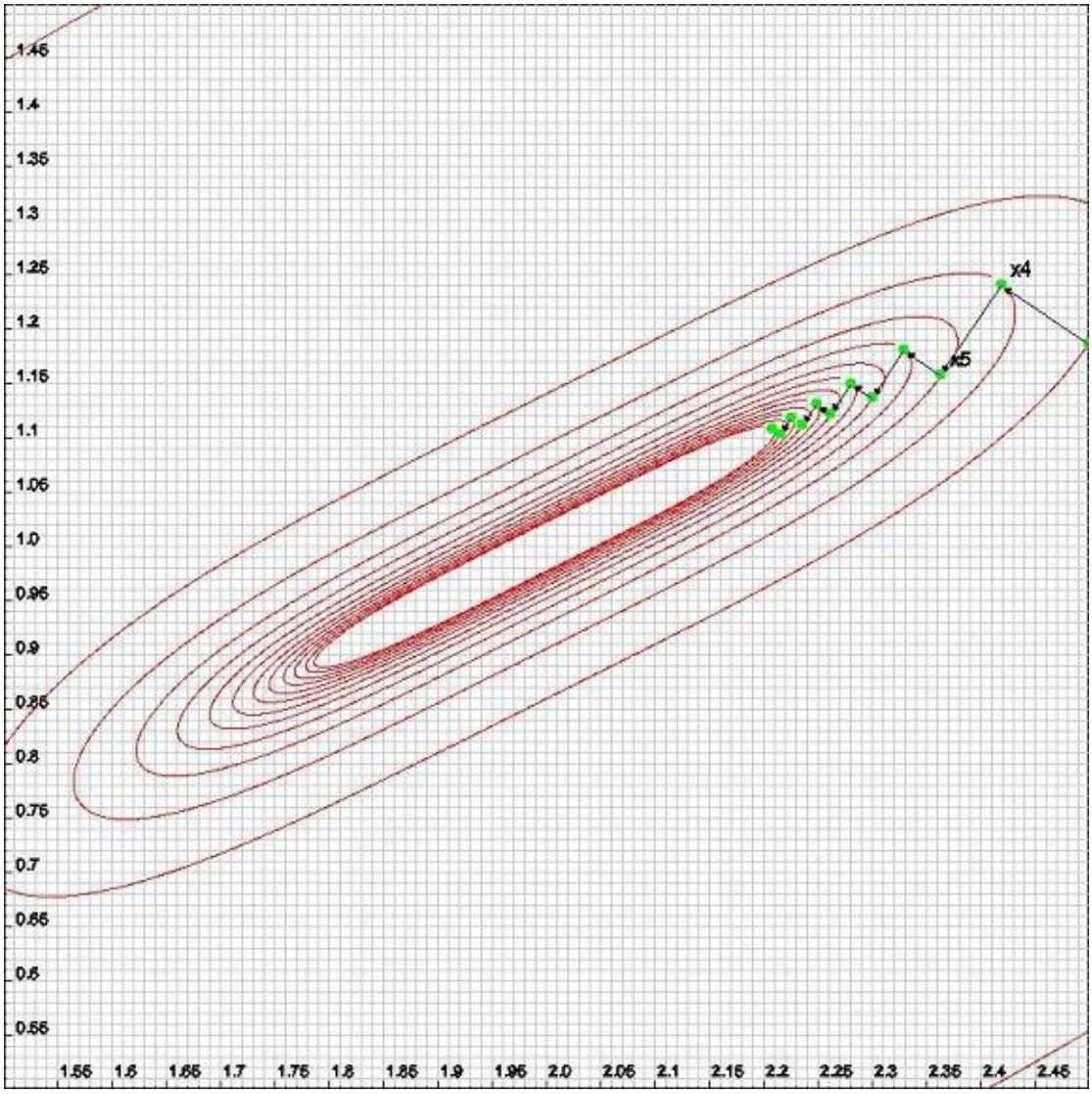

FIG. 3.2 – *illustration de la convergence lente de la méthode du gradient pour le problème P2 en partant depuis (0,4).*

durant les premières itérations, la méthode du gradient s'est rapidement approché du minimum (2, 1), se déplaçant du point (0, 4) en (2.3, 1.14) en 7 itérations ; cependant, ensuite, 27 itérations lui ont été nécessaires pour se déplacer de (2.3, 1.14) à (2.13, 1.07). La méthode a donc convergé rapidement au début, avant de tendre ensuite très lentement vers le minimum. Ceci est typiquement dû au phénomène de zigzag que nous avons déjà décrit.

La méthode de Newton, elle, se rapproche plus lentement du minimum durant les premières itérations que la méthode du gradient (jusqu'à l'itération 3 ou 4) ; cependant elle converge ensuite beaucoup plus vite, car les directions générées ne tendent pas à être orthogonales à la direction menant au minimum.

Les figures 3.2 et 3.3 illustrent les comportements de ces deux méthodes.

Si l'on affine la condition d'arrêt en spécifiant par exemple $|\nabla f(x^k)| < 0.001$, en partant toujours de (0,4), la méthode du gradient a besoin de 150 itérations pour terminer au point

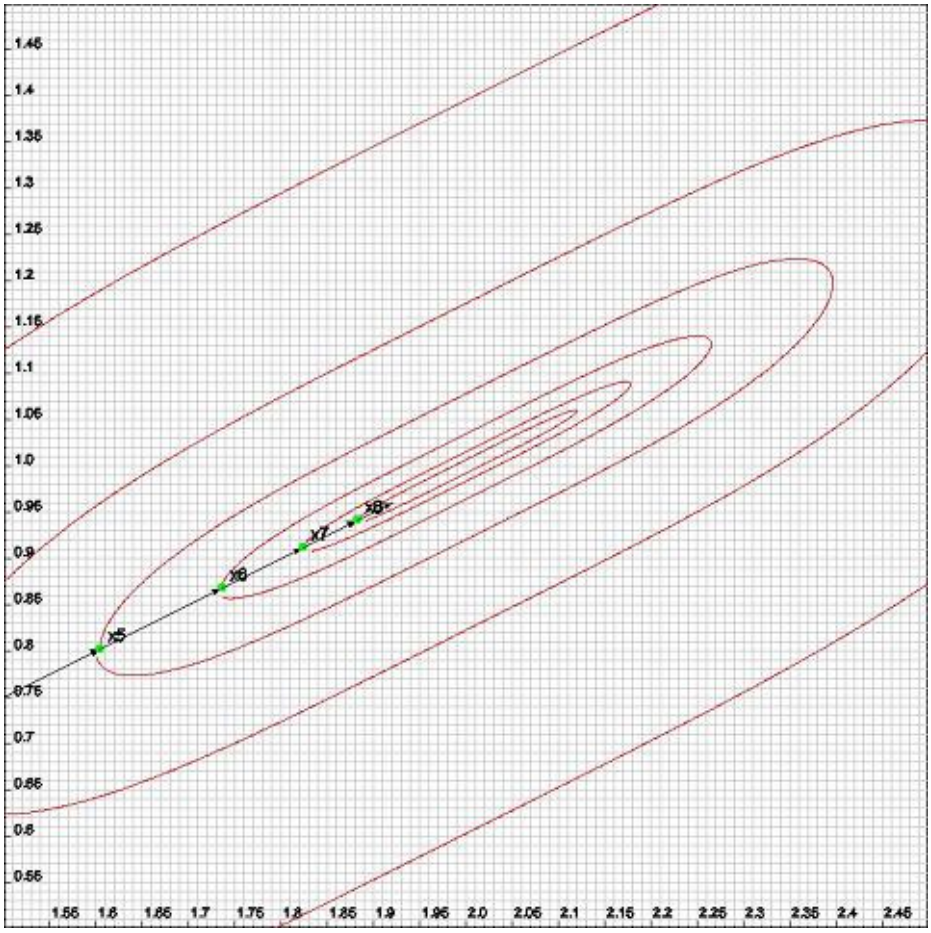

FIG. 3.3 – *illustration de la convergence plus rapide de la méthode de Newton pour le même problème.*

(2.06, 1.03). La méthode de Newton, sous les mêmes conditions, ne nécessite que 10 itérations et s'arrête en (1.95, 0.97). Newton donne donc des résultats considérablement meilleurs pour ce problème, en tout cas avec ce point de départ proche du minimum. Cela montre toutes les limites de la méthode du gradient sous sa forme classique.

Reproduisons l'expérience avec un point initial éloigné du minimum, par exemple (10^7, $-2,5 \cdot 10^6$) (la condition d'arrêt est $|\nabla f(x^k)| < 0.01$). La méthode du gradient se termine après 1269 itérations alors que la méthode de Newton n'a besoin que de 46 itérations.

Signalons tout de même, à la décharge de la méthode du gradient, qu'une itération de la méthode de Newton se révèle plus gourmande en calculs et plus lente à l'exécution qu'une itération de la méthode du gradient à cause de l'évaluation de l'inverse de la matrice hessienne.

Nous pouvons recommencer l'expérience en choisissant un point initial plus éloigné encore, par exemple ($5 \cdot 10^{10}$, 10^9). La méthode du gradient a alors besoin de 36'757 itérations pour terminer en (1.94, 0.97). La méthode de Newton, elle se termine après 69 itérations seulement au point (2.05, 1.03). Le constat est donc sans appel.

Nous pouvions nous attendre, à priori, à ce que la méthode du gradient donne de meilleurs résultats, lorsque le point initial est éloigné, entre autres grâce à la minimisation monodimentionnelle effectuée le long de d^k. L'expérience précédente montre que ce n'est pas le cas (bien entendu, on ne peut néanmoins pas généraliser avant d'avoir répété ce type d'expériences à de nombreuses reprises et avec différents problèmes). L'avantage que la minimsation monodimentionnelle donne à la méthode du gradient est dominé par le fardeau que constitue le zigzag : avec cette dernière, suite à une quarantaine d'itérations, la composante x_1 a diminué et est de l'ordre de 10^4, cependant la composante x_2 est toujours stable et de l'ordre de 10^9, ce qui montre que la méthode s'est mise à "zigzaguer" très tôt, expliquant la mauvaise performance. Pour la méthode de Newton, le point trouvé à l'itération 40 est (6784.83, 3392.41). Ainsi, dès les premières itération, la méthode de Newton a progressé considérablement plus vite que la méthode du gradient. On soulignera cependant deux choses : tout d'abord, le comportement de la méthode du gradient est fortement influencé par le point initial lui-même (on ne parle pas ici de sa distance au minimum). Il est possible que, exécutée depuis un autre point tout aussi éloigné, le nombre d'itérations nécessité par la méthode du gradient eût été bien moindre (il est par exemple intéressant de constater que démarrer la méthode en ($5 \cdot 10^{10}$, $-1 \cdot 10^9$) ramène le nombre d'itérations nécessaire à 22'999). De plus, il se peut que cette fonction soit particulièrement bien approchable par une fonction quadratique.

Exécutons maintenant l'algorithme de Nelder et Mead pour résoudre ce problème, en partant d'un simplexe arbitraire non loin du minimum (le simplexe initial est formé par (0,4), (-3,-2) et (-4,7) et la condition d'arrêt est $|\nabla f(x^k)| < 0.01$). 18 itérations sont nécessaires et l'algorithme se termine en (2.11, 1.05). Si l'on remplace la valeur de la condition d'arrêt par 0.001, 29 itérations sont nécessaire et l'algorithme se termine en (2,1). Si cet algorithme donne en général de bons résultats, il faut cependant éviter de choisir un simplexe initial dont les points seraient trop éloignés.

Chapitre 4

Méthodes itératives d'optimisation avec contraintes

Nous considérons ici le problème

$$\begin{aligned} &\text{minimiser } f(x) \\ &\text{sous contrainte } x \in X \end{aligned}$$

où X est défini par une collection d'inégalités $g_i(x) \leq 0$ et d'égalités $h_j(x) = 0$, c'est-à-dire

$$X = \{\ x \mid g_i(x) \leq 0,\ h_j(x) = 0, \quad \text{avec } i = 1, ..., m \text{ et } j = 1, ..., r\}.$$

Nous décrivons les principales méthodes itérative implémentées pour ce problème. Il s'agira des classes de méthodes suivantes :

1. Les méthodes de directions admissibles
2. Les méthodes de plans sécants
3. Les méthodes de pénalité intérieure
4. Les méthodes de pénalité extérieure

Les champs d'application de ces méthodes (soit les hypothèses sous lesquelles elles s'appliquent) seront notamment mentionnés et elles seront analysées du point de vue de leur convergence. Après les avoir utilisées en pratique, nous comparerons leur performances. Enfin, des stratégies destinées à fournir aux algorithmes un point initial admissible en partant d'un point quelconque de $\mathbb{R}^n$ seront discutées.

Remarque : toute contrainte sous forme d'égalité peut être exprimée de manière équivalente par deux inégalités :

$$h_j(x) = 0$$

peut être exprimé sous la forme

$$\begin{aligned} h_j(x) &\leq 0 \\ -h_j(x) &\leq 0. \end{aligned}$$

Au travers des sections suivantes, nous admettrons donc que X n'est défini que part une collection de m inégalité $g_i(x) \leq 0$, puisque cela n'induit aucune perte de généralité. Nous agirons de la sorte pour toutes les méthodes sauf la méthode de barrière et la méthode de Zoutendijk pour les contraintes non linéaires, qui, nous verrons pourquoi, ne sont pas en mesure de traiter de contraintes sous forme d'égalités.

4.1 Les méthodes de directions admissibles

Cette classe de méthodes résout un problème de minimisation non linéaire en se déplaçant d'un point de X vers un autre de ses points au coût inférieur. Elles fonctionnent selon le principe suivant : étant donné un élément x^k de X, une direction d^k est générée telle que pour un $\alpha^k > 0$ et suffisamment petit, les propriétés suivantes sont assurées :

1. $x^k + \alpha^k d^k$ appartient toujours à X,
2. $f(x^k + \alpha^k d^k)$ est inférieur à $f(x^k)$.

Une fois d^k déterminée, α^k s'obtient par minimisation monodimentionnelle pour que le déplacement dans la direction d^k soit optimal, mais cette fois-ci il est nécessaire d'imposer une borne supérieure sur la valeur de α^k afin de ne pas sortir de X. Cela définit le nouveau point x^{k+1} et le processus est recommencé.

Il existe plusieurs stratégies pour la résolution du sous-problème consistant à déterminer d^k. Comme nous allons le voir, il peut être exprimé sous forme d'un programme linéaire. Nous décrivons deux méthodes de directions admissibles qui ont été implémentées, appelées méthodes de Frank et Wolfe et de Zoutendijk.

4.1.1 La méthode de Frank et Wolfe

Description de la méthode

Cette méthode s'applique si les contraintes sont linéaires et si X est borné. Une manière simple de générer une direction d^k satisfaisant la condition de descente $\nabla f(x^k) \cdot d^k < 0$, si l'on pose $d^k = \bar{x}^k - x^k$, est de minimiser la dérivée directionnelle de f dans la direction d^k, comme c'est le cas pour la méthode du gradient ; mais il faut en plus prendre garde de pas sortir de l'ensemble des solutions admissibles (le point $\bar{x}^k$ généré doit lui-même appartenir à X, de sorte que le point $x^k + d^k$ soit lui-même admissible). Le sous-problème de recherche de d^k peut être formulé ainsi :

$$\begin{aligned} &\text{minimiser } \nabla f(x^k) \cdot (x - x^k) \\ &\text{sous contrainte } x \in X \end{aligned}$$

et nous pouvons obtenir $\bar{x}^k$ comme la solution optimale de ce sous-problème. Celui-ci est un programme linéaire (les contraintes du problème de départ le sont elles-même) et peut être résolu par l'algorithme du simplexe. Selon un théorème connu, sa solution optimale $\bar{x}^k$ se trouvera alors en l'un des points extrêmes du domaine X, de sorte que l'optimisation monodimentionnelle le long de d^k devra être faite sous la restriction $0 \leq \alpha^k \leq 1$. La méthode se termine si la direction générée est égale au vecteur nul.

Convergence de la méthode

Le résultat suivant peut être démontré :

Théorème 4.1.1. Considérons le problème de minimisation de $f(x)$ sous contraintes $x \in X$ où X est un polytope borné. Alors la méthode de Frank et Wolfe démarrée depuis un point initial admissible converge vers un point satisfaisant la condition de Kuhn-Tucker.

La méthode converge vers un point de Kuhn-Tucker, mais pas nécessairement de manière finie car elle peu être sujette au zigzag, à l'instar de la méthode du gradient (nous en reparlerons lors de son utilisation pratique).

4.1.2 La méthode de Zoutendijk

La méthode de Zoutendijk fonctionne selon le même schéma : à chaque itération, elle génère une direction de descente admissible et ensuite minimise f le long de cette direction. A l'image de la méthode de Frank et Wolfe, le sous-programme de recherche d'une telle direction est linéaire. Nous commençons par décrire la méthode de Zoutendijk applicable si les contraintes sont linéaires, puis celle-ci suscitera des modifications si elle ne le sont pas afin de garantir que des directions non admissibles ne pourront être générées si la méthode se trouve en un point où une contrainte non linéaire est active.

Cas des contraintes linéaires

La méthode de Zoutendijk cherche elle aussi à trouver, à chaque itération, la direction d^k minimisant la dérivée directionnelle de f en x^k. Les variables apparaissant dans la fonction objectif du sous-programme linéaire sont les composantes de d^k, que l'on norme afin d'empêcher le problème d'être non borné. Si le point x^k courant est situé à l'intérieur de X, le problème de recherche de la direction optimale d peut s'écrire :

$$\begin{aligned} &\text{minimiser } \nabla f(x^k) \cdot d \\ &\text{sous contraintes} \quad d_i \leq 1 \qquad \forall i \in \{1, ..., n\}, \\ &\qquad\qquad\qquad\ d_i \geq -1 \qquad \forall i \in \{1, ..., n\}. \end{aligned}$$

Si x^k se trouve maintenant sur la frontière du domaine, une ou plusieurs contraintes du problème original y sont actives. Il faut donc ajouter au programme linéaire précédent des contraintes excluant les directions menant uniquement à des points non admissibles. Cela est fait simplement en faisant intervenir les gradients des contraintes actives. Soit I l'ensemble des contraintes actives en x^k : $I = \{\ i \mid g_i(x^k) = 0\}$. Le problème de recherche de direction final est :

$$\begin{aligned} &\text{minimiser } \nabla f(x^k) \cdot d \\ &\text{sous contraintes} \quad \nabla g_i(x^k) \cdot d \leq 0 \qquad \forall i \in I, \\ &\qquad\qquad\qquad\ d_i \leq 1 \qquad \forall i \in \{1, ..., n\}, \\ &\qquad\qquad\qquad\ d_i \geq -1 \qquad \forall i \in \{1, ..., n\}. \end{aligned}$$

Un résultat intéressant est donné par le théorème suivant :

Théorème 4.1.2. Les conditions nécessaires de Kuhn-Tucker sont satisfaites en x^k si et seulement si la valeur optimale de la fonction objectif du programme linéaire précédent est égale à zéro.

Ainsi, la méthode de Zoutendijk se termine si la valeur optimale de la fonction objectif de ce programme linéaire est nulle, car elle se trouve alors en un point x^k qui satisfait les conditions de Kuhn-Tucker.

Cas des contraintes non linéaires

Si, au point courant x^k, une contrainte non linéaire g_i est active, le problème de recherche d'une direction présenté au paragraphe précédent peut aboutir au choix d'une direction non admissible. En effet, un vecteur d satisfaisant $\nabla g_i(x^k) \cdot d = 0$ est tangent à la courbe $g_i(x) = 0$. En raison de la non linéarité de g_i, il se peut que tout déplacement dans cette direction conduise en

un point non admissible. Cette approche doit donc être modifiée pour pouvoir prendre en compte ce type de contraintes. Nous énonçons ci-dessous un théorème qui permettra de surmonter cette difficulté :

Théorème 4.1.3. Considérons le problème de minimisation de $f(x)$ sous contraintes $g_i \leq 0$ pour $i = 1, ..., m$. Soient x^k une solution admissible et I l'ensemble des contraintes actives en $x^k : I = \{ i \mid g_i(x^k) = 0\}$. Supposons, de plus, que f et les g_i sont continues et continuellement différentiables. Si $\nabla f(x^k) \cdot d < 0$ et $\nabla g_i(x^k) \cdot d < 0 \ \forall i \in I$, alors d est une direction de descente admissible de f en x^k.

Ce résultat ne fait que confirmer ce à quoi nous nous attendions, c'est-à-dire que l'inégalité stricte $\nabla g_i(x^k) \cdot d < 0$ est nécessaire pour garantir la génération d'une direction admissible (alors que l'inégalité stricte $\nabla f(x^k) \cdot d < 0$ doit être satisfaite pour qu'elle soit de descente).

Pour trouver un vecteur d satisfaisant ces deux inégalités strictes, une possibilité est de minimiser le maximum de la valeur $\nabla f(x^k) \cdot d$ et des valeurs $\nabla g_i(x^k) \cdot d$ pour $i \in I$. En dénotant ce minimum par z et en introduisant les restrictions interdisant au problème d'être non borné, nous obtenons le problème suivant :

$$
\begin{array}{ll}
\text{minimiser} & z \\
\text{sous contraintes} & \nabla f(x^k) \cdot d - z \leq 0, \\
& \nabla g_i(x^k) \cdot d - z \leq 0 \qquad \forall i \in I, \\
& d_i \leq 1 \qquad \forall i \in \{1, ..., n\}, \\
& d_i \geq -1 \qquad \forall i \in \{1, ..., n\}.
\end{array}
$$

Si la valeur optimale de z est strictement inférieure à zéro, alors d est manifestement une direction de descente admissible. Si elle vaut zéro, x^k satisfait la condition de Fritz John, comme l'établit le théorème 4.1.4.

Théorème 4.1.4. Considérons le problème de minimisation de $f(x)$ sous contraintes $g_i \leq 0$ pour $i = 1, ..., m$. Soient x^k une solution admissible et $I = \{ i \mid g_i(x^k) = 0\}$. Alors x^k satisfait la condition de Fritz John si et seulement si la valeur optimale de la fonction objectif du programme linéaire précédent est égale à zéro.

Convergence de la méthode

En général, la convergence de la méthode de Zoutendijk n'est pas garantie. Un contre-exemple, attribué à Wolfe, a montré qu'elle pouvait ne pas converger vers un point satisfaisant les conditions de Kuhn-Tucker.

Observons qu'avec l'emploi de cette méthode, nous ne pouvons pas traiter de contraintes non linéaires qui soient sous forme d'égalités, car il ne serait pas possible de générer une direction de déplacement qui n'amène à sortir du domaine. Cette difficulté peut être surmontée en introduisant des mouvement correctifs destinés à revenir dans la région admissible, mais de telles stratégies n'ont pas été implémentées dans ce projet, nous ne les détaillerons donc pas ici.

4.2 Les méthodes de plans sécants

Les méthodes de plans sécants sont applicables si X est fermé, si les contraintes qui le définissent sont convexes (X l'est donc également) et si f est convexe. Si f est linéaire, le principe de ces méthodes est "d'enfermer" l'ensemble des solutions admissibles dans un polytope

P formé d'au moins $n+1$ hyperplans (c'est le nombre minimal d'hyperplans que doit comporter le polytope initial, au début de l'exécution). P définit alors un nouvel ensemble de solutions admissibles et le problème consistant à minimiser f sur P est un programme linéaire. Si sa solution optimale appartient à X, l'exécution est terminée car cette solution est également celle du problème original. Sinon, un nouvel hyperplan est introduit réduisant P à un nouveau polytope. Celui-ci doit répondre à deux exigences : il ne doit tout d'abord éliminer aucune solution admissible du problème original ; ensuite, la solution optimale du programme linéaire résolu à l'itération précédente doit devenir non admissible. Puis, le processus est reconduit jusqu'à satisfaire la condition d'arrêt.

Remarque : il n'y a pas de perte de généralité à supposer que f est linéaire, car si ce n'est pas le cas, les méthodes de plans sécants peuvent être appliquées à un problème auxiliaire dont la fonction objectif est linéaire et comporte une contrainte supplémentaire, de telle sorte que la solution optimale de ce problème auxiliaire soit la même que celle du problème original.

Parmi les stratégies possibles pour sélectionner un nouvel hyperplan à chaque itération, l'une a été proposée par Kelley. Décrivons la méthode de Kelley qui a été implémentée au cours de ce projet.

4.2.1 La méthode de Kelley

Description de la méthode

En plus des hypothèses formulées au début de la section 4.2, nous admettons donc que f est linéaire. X est initialement enfermé dans un ensemble de contraintes linéaires qui sont habituellement choisies comme de simples bornes sur les variables. Le point obtenu à la première itération x^1 est la solution du programme linéaire correspondant. Si x^1 n'est pas une solution admissible du problème de départ, il existe au moins un i tel que $g_i(x) > 0$. Nous pouvons alors choisir s tel que $g_s(x^k) = \max_{1 \leq i \leq m} \ g_i(x^k) > 0$. Une nouvelle contrainte linéaire est ensuite ajoutée, donnée par l'approximation linéaire de $g_s(x)$ au voisinage de x^k :

$$g_s(x^k) + \nabla g_s(x^k) \cdot (x - x^k) \leq 0.$$

Remarquons que ce nouvel hyperplan coupe x^k, qui ne satisfait pas la relation précédente, du domaine admissible original qui est inclus dans le demi-espace défini par cette relation. La solution du programme linéaire précédent, auquel cette contrainte a été ajoutée, est le nouveau point x^{k+1}, puis le processus recommence jusqu'à satisfaire la condition d'arrêt.

Convergence de la méthode

Le théorème 4.2.1 stipule que la méthode de Kelley converge :

Théorème 4.2.1. Soit une séquence $\{x^k\}$ générée par la méthode de kelley sous les hypothèses énoncées à la section 4.2. Alors tout point limite de $\{x^k\}$ est la solution optimale globale du problème original.

4.3 Les méthodes de pénalité intérieure (ou méthodes de barrière)

Nous allons maintenant décrire les méthodes de pénalité. Cette section traite des méthodes de pénalité intérieure, aussi appelées *méthodes de barrière*, alors que les méthodes de pénalités extérieure seront détaillées à la section suivante. Le principe de ces méthodes réside dans la transformation d'un problème contraint en une séquence de problèmes sans contraintes, en

ajoutant au coût une pénalité en cas de violation de celles-ci. Un tel sous-problème est résolu à chaque itération d'une méthode de pénalité. L'appellation "pénalité intérieure" est employée car le minimum est approché depuis l'intérieur de X ; les méthodes de pénalité extérieure, elles, approchent le minimum depuis l'extérieur de cet ensemble.

Les méthodes de barrière s'appliquent aux problèmes dont l'ensemble admissible X est défini uniquement par une collection d'inégalités :

$$\begin{aligned} &\text{minimiser } f(x) \\ &\text{sous contraintes } g_i(x) \leq 0, \qquad i = 1, ..., m \end{aligned}$$

où f et les g_i sont des fonction de $\mathbb{R}^n$ dans $\mathbb{R}$ et sont continues. En effet, ces méthodes utilisent des fonctions dites de barrière, définies uniquement à l'intérieur de X. Si des contraintes sous forme d'égalités étaient introduites, l'intérieur de cet ensemble, c'est-à-dire son sous-ensemble tel qu'en chacun de ses points aucune contrainte n'est active, serait clairement vide. C'est donc le domaine de définition de la fonction de barrière qui serait vide rendant l'utilisation de la méthode impossible.

L'*intérieur* de l'ensemble X défini par les g_i est le suivant :

$$X_I = \{\ x \mid g_i(x) < 0,\ \forall i \in \{1, ..., m\}\ \}.$$

La *fonction de barrière*, notée $B(x)$, est ajoutére au coût $f(x)$; elle est continue sur X_I et sa valeur tend vers l'infini lorsque la frontière de X est approchée par l'intérieur, c'est-à-dire lorsque l'un des $g_i(x)$ approche zéro par les valeurs négatives. Une itération de la méthode consiste ensuite minimiser la fonction $f(x) + \epsilon B(x)$ (où ϵ est un paramètre réel strictement positif) à l'aide d'algorithmes de minimisation directe, comme par exemple ceux décrits au chapitre 3, une fonction $B(x)$ et un ϵ convenablement choisis assurant que cette minimisation ne puisse nous mener à des points situés hors de X_I. La suite du processus consiste à réduire progressivement ϵ afin de diminuer la pénalité et autoriser les algorithmes de minimisation directe à se rapprocher peu à peu de la frontière de X.

Les fonctions de barrière les plus répandues sont les suivantes :

La logarithmique :

$$B(x) = -\sum_{i=1}^{m} \ln(-g_i(x)).$$

L'inverse :

$$B(x) = -\sum_{i=1}^{m} \frac{1}{g_i(x)}.$$

Il est important de noter que, si tous les g_i sont convexes, ces deux fonctions de barrière le sont également.

Remarquons qu'une nécessité, pour pouvoir appliquer une telle méthode, est de disposer d'un point initial situé à l'intérieur de X. La méthode de barrière est définie en introduisant la séquence de paramètres $\{\epsilon^k\},\ k = 0, 1, ...$ avec $0 < \epsilon^{k+1} < \epsilon^k$ et $\epsilon^k \to 0$ lorsque $k \to \infty$.

Une itération de celle-ci consiste à déterminer

$$x^k = \arg\min_{x \in X_I} \{f(x) + \epsilon^k B(x)\}.$$

Le fait que $B(x)$ ne soit défini que dans l'intérieur X_I et tende vers l'infini au fur et à mesure que l'on se rapproche des bords de X assure que, même avec un algorithme de minimisation

directe, le point obtenu à chaque itération appartienne lui aussi à X_I. Le fait que ϵ^k tende vers zéro implique que le terme $f(x) + \epsilon^k B(x)$ tende vers $f(x)$ lorsque k tend vers l'infini.

Si l'on admet que la définition de x^k est celle énoncée ci-dessus, il est possible de démontrer le théorème suivant :

Théorème 4.3.1. Tout point limite d'une séquence $\{x^k\}$ générée par une méthode de barrière est un minimum global du problème contraint original.

Cependant, en pratique, la définition précédente de x^k implique que nous devions disposer d'un algorithme de minimisation directe capable de trouver le minimum global de $f(x)+\epsilon^k B(x)$ à chaque itération. Or, ce n'est pas le cas des algorithmes présentés au chapitre précédents, notamment les algorithmes du gradient, de Newton, ou encore du simplexe, qui peuvent être aussi bien attirés par des minima locaux que globaux. Si nous utilisons de tels algorithmes pour la résolution des sous-problème, comme c'est notamment le cas du logiciel développé durant ce projet, il nous est impossible d'assurer qu'à chaque itération x^k soit le minimum global de $f(x)+\epsilon^k B(x)$ sur X_I (du moins si f et les g_i ne sont pas convexes). Si x^k est réduit à n'en être qu'un minimum local, le théorème précédent ne peut être établi.

Si l'algorithme utilisé pour l'optimisation sans contraintes est la méthode de Newton, celle-ci peut être attirée par des maxima locaux aussi bien que part des minima. Dans un tel cas, il peut arriver que la méthode de barrière converge globalement vers un maximum et non un minimum local, ou encore vers un point stationnaire.

Signalons que si f et les g_i sont convexe, $f(x)+\epsilon B(x)$ l'est également, car elle est la somme de fonctions convexes. Tout point stationnaire d'une telle fonction étant un minimum global (voir la section 2.2), le théorème 4.3.1 s'applique et nous avons la garantie que la méthode de barrière convergera vers le minimum global du problème contraint original, et ce même si l'algorithme de minimisation directe utilisé n'est voué qu'à approcher des minima locaux.

Bien entendu, le comportement de la méthode dépend fortement du choix du paramètre ϵ^0 initial et du facteur, disons β, satisfaisant $0 < \beta < 1$, utilisé pour décroître ϵ^k à chaque itération par la formule $\epsilon^{k+1} = \beta\epsilon^k$. Il n'existe a pas de règle universelle permettant d'obtenir un bon choix de ϵ^0 et de β. Cela dépend fortement du problème à résoudre ainsi que du point initial x^0 (celui-ci se trouve-t-il ou non à proximité de la frontière du domaine ?). L'utilisateur d'une méthode de barrière sera souvent condamné à exécuter la méthode plusieurs fois avec différentes valeurs de ces paramètres jusqu'à obtenir une convergence satisfaisante.

Les faits suivants peuvent néanmoins être relevés : si ϵ^k est trop petit, et à plus forte raison si x^k est proche des bords du domaine, le terme de barrière peut se révéler trop faible ne parvenant pas à empêcher une sortie de X (il faut donc prendre garde, durant l'exécution, et vérifier que l'on ne sorte pas de cet ensemble, et, si cela est tout de même le cas, recommencer avec un ϵ^0 ou un β plus grand). Dans le cas contraire, si ϵ^k est trop grand, l'algorithme de minimisation directe ne pourra s'approcher suffisamment des bords du domaine, conduisant à une convergence globale lente. Ces paramètres doivent donc être soigneusement choisis d'après le problème et le point initial.

Remarque : en dehors de x^0, les points obtenus à l'itération précédente sont, à chaque itération, utilisés comme points de départ de l'algorithme de minimisation directe (cela est valable pour les deux types de méthodes de pénalité).

4.4 Les méthodes de pénalité extérieure

Au contraire de celles que nous venons de décrire, les méthodes de pénalité extérieures cherchent à approcher le minimum depuis l'extérieur de X. Au coût $f(x)$ est ajouté une *fonction de pénalité extérieure* $P(x)$ dont la valeur est égale à zéro si x est admissible et supérieure à

zéro s'il ne l'est pas. L'ajout de cette fonction a pour seul but de pénaliser le coût en cas de violation d'une ou de plusieurs contraintes. Dans ce projet, la fonction de pénalité quadratique est utilisée, c'est-à-dire que $P(x)$ est définie ainsi :

$$P(x) = \sum_{i=1}^{m} (g_i(x))^2 \; u_i(g_i(x))$$

où

$$u_i(g_i(x)) = \begin{cases} 0 & \text{si } g_i(x) \leq 0, \\ 1 & \text{si } g_i(x) > 0. \end{cases}$$

La fonction $u_i(g_i(x))$ sert à ignorer la pénalité si la contrainte correspondante est satisfaite. De façon analogue à la méthode de barrière, nous introduisons un paramètre μ qui permet d'amplifier ou de diminuer sa valeur et, à chaque itération, le sous-problème à résoudre à l'aide d'une méthode de minimisation directe sera de la forme suivante :

$$\begin{aligned} &\text{minimiser } f(x) + \mu P(x) \\ &\text{avec } x \in \mathbb{R}^n \end{aligned}$$

Lorsque μ est grand, une plus grande importance est attachée à l'admissibilité (au sens du problème contraint original), ce qui suggère que la méthode doit commencer avec un μ qui n'ait pas une valeur trop élevée (afin d'éviter une terminaison prématurée en un point certes admissible mais qui pourrait se trouver loin de l'optimum), puis augmenter celui-ci progressivement pour s'approcher de l'ensemble admissible du problème original. Ainsi la séquence $\{\mu^k\}$ utilisée devra être telle que $0 < \mu^k < \mu^{k+1}$, avec $\mu^k \to \infty$ lorsque $k \to \infty$. Un paramètre additionnel $\beta > 1$ est utilisé pour générer $\mu^{k+1} = \beta\mu^k$. A chaque itération, la méthode obtient donc x^k de la manière suivante :

$$x^k = \arg\min_{x \in \mathbb{R}^n} \{f(x) + \mu^k P(x)\}.$$

Le comportement de la méthode de pénalité extérieure est très similaire à celui de la méthode de barrière. En particulier, des résultats théoriques équivalents peuvent être établi et assurer sa convergence. Les considérations plus pratiques, principalement lorsque μ croît, sont également similaires à celles données pour la méthodes de barrière (voir les sections 4.3 et 4.6), la seule différence notable étant que la méthode de pénalité extérieure ne nécessite aucunement un point initial intérieur à X.

4.5 Comment trouver un point initial admissible ?

La plupart des méthodes que nous venons de décrire doivent démarrer d'un point admissible (la méthode de barrière est même plus exigeante car elle doit partir depuis un point admissible intérieur). Pour certains problèmes, il se peut qu'un tel point ne soit pas immédiatement disponible et ne soit pas facile à trouver, notamment si les contraintes ne sont pas linéaires. Il est important de pouvoir disposer de stratégies permettant de trouver un tel point à partir d'un point quelconque. Considérons les méthodes de directions admissibles, qui doivent partir d'un point satisfaisant les contraintes non nécessairement intérieur et décrivons une méthode pour trouver un tel point.

Soit le problème de minimisation de $f(x)$ sous contraintes $g_i(x) \leq 0$ avec $i = 1, ..., m$. Nous choisissons un point $\bar{x}$ quelconque. On définit I, l'ensemble des contraintes satisfaites en $\bar{x}$: $I = \{\ i \mid g_i(\bar{x}) \leq 0\}$. Nous avons donc $g_i(\bar{x}) > 0\ \forall i \notin I$. Penchons-nous sur le problème suivant :

$$\begin{aligned} &\text{minimiser } \sum_{i \notin I} y_i \\ &\text{sous contraintes } g_i(x) \leq 0 \qquad \forall i \in I, \\ &\qquad\qquad\qquad\quad g_i(x) - y_i \leq 0 \qquad \forall i \notin I, \\ &\qquad\qquad\qquad\quad y_i \geq 0 \qquad \forall i \notin I. \end{aligned}$$

Il est possible de montrer qu'une solution admissible du problème original existe si et seulement si la valeur objective optimale du problème ci-dessus est égale à zéro. Soit y un vecteur dont les composantes sont y_i pour $i \notin I$. Le problème ci-dessus peut être résolu par une méthode de directions admissibles en partant du point $(\bar{x}, \bar{y})$ où $\bar{y}_i = g_i(\bar{x})$ pour $i \notin I$. Lorsque celle-ci se termine, une solution admissible du problème original est obtenue. Alors, une méthode de directions admissibles peut être utilisée à nouveau pour résoudre le problème original.

4.6 Etude comparative des méthodes d'optimisation avec contraintes

Les méthodes de pénalité

Les méthodes de directions admissibles atteignent les points situés sur la frontière du domaine, et donc obtiennent généralement une approximation plus précise d'un minimum local que les méthodes de pénalité. Bien que la méthode de barrière soit très bien supportée par la théorie et nécessite peu de restrictions sur le problème pour assurer une convergence théorique, elle souffre de faiblesses dans le cadre de l'utilisation pratique, particulièrement lorsque la frontière de la région admissible est approchée :

- Il n'y a pas de technique générale pour le choix du paramètre ϵ^0 et du facteur de réduction, ce qui implique qu'il faille souvent redémarrer la méthode plusieurs fois.
- Lorsque $\{\epsilon_k\}$ est très proche de zéro, l'arrondissement des valeurs dû au calcul en nombre de chiffres limités que l'ordinateur utilise entraîne la propagation des erreurs dûes à celui-ci.
- Lorsque la méthode approche l'extrêmité du domaine, un problème est posé, dû aux algorithmes de minimisation directe utilisés. L'on se convaincra aisément du fait que le comportement d'une méthode de barrière dépend fortement du choix de cet algorithme. Ceux-ci supposent habituellement que leur fonction objectif est continue et définie sur tout $\mathbb{R}^n$, ce qui n'est pas le cas avec une fonction de barrière. Lorque le bord du domaine est approché, la discontinuité se trouve dans le voisinge immédiat du point de départ de la procédure.

Pour ces raisons, nous devons disposer d'une arithmétique à double précision afin de limiter au maximum les erreurs dûes aux arrondis. Dans notre cas, un choix doit être opéré parmi les algorithmes présentés au chapitre 3 pour la résolution des sous-problèmes. Il est préférable d'utiliser pour cela la méthode de Newton, qui propose une convergence plus rapide comme nous l'avons vu que la méthode du gradient, et qui offre des garanties quant à sa convergence contrairement à la méthode de Nelder et Mead.

Observons que la méthode de Newton est précisément sensible à l'un des défauts majeurs de la méthode de barrière, qui est le conditionnement de plus en plus mauvais de la matrice hessienne de la fonction objectif à mesure que la borne est approchée (rappelons qu'une matrice mal conditionnée est une matrice "presque singulière", dans le sens où elle pourrait devenir

singulière si l'un de ses éléments était faiblement modifié). De nombreux auteur ont montré que le nombre de conditionnement de cette matrice augmente proportionnellement à l'inverse de ϵ, de sorte que la matrice est de plus en plus mal conditionnée à mesure que ϵ approche 0. Elle tend alors vers la singularité, ce qui contribue a dégrader la performance de la méthode de Newton employée pour la minimisation directe.

Ainsi, on s'aperçoit qu'en vue d'une utilisation pratique les méthodes de barrière ne sont pas exemptes de défauts. Néanmoins, elles présentent aussi des avantages certains sur les méthodes de directions admissibles. Elles se révèlent tout d'abord particulièrement efficaces lorsque les contraintes sont non linéaires. En effet, si les contraintes linéaire peuvent être prises en charge avec succès par de nombreux algorithmes dont les méthodes de directions admissibles, il en est autrement des contraintes non linéaires (nous avons vu qu'elles avaient plus de difficulté à les traiter). Au contraire de ces dernières qui traitent les contraintes non linéaires explicitement, les méthodes de pénalité évitent la tâche compliquée et coûteuse consistant à se déplacer le long de la frontière de l'ensemble admissible en prenant garde à ne pas la franchir (ou en effectuant des mouvements correctifs après en être sorti). De plus, aucune hypothèse particulière ni sur la fonction objectif ni sur les contraintes n'est nécessaire pour que la méthode puisse être appliquée (comme c'est le cas par exemple pour la méthode de Kelley, qui requiert des contraintes convexes).

Les méthodes de directions admissibles

Les méthodes de directions admissibles qui ont été implémentées (celles de Frank et Wolfe et de Zoutendijk) se basent sur le gradient de la fonction objectif pour déterminer une direction de mouvement ; on ne peut donc pas attendre de ces méthodes qu'elle convergent vers des minima globaux ; elle seront attirée par tout type de minimum, local ou global. Obsevons que la méthode de Frank et Wolfe puisse dans certain cas également tomber dans le cas d'une convergente très lente en zigzag. En effet, le point $\bar{x}^k$ determiné à chaque itération afin de produire une direction est typiquement un point extrême du polytope formé par les contraintes (supposées linéaires, rappelons-le). Il est donc possible que la direction d^k obtenue soit presque orthogonale à la direction menant au minimum (voir la figure 4.5).

Nous allons maintenant résoudre des problèmes à l'aide des diverses méthodes et critiquer les résultats obtenus. Soit P_1 :

$$\begin{aligned}
\text{minimiser } & -(x_1-4.2)^2-(x_2-1.9)^2 \\
\text{sous contraintes } & -x_1+x_2 \leq 3 \\
& x_1+x_2 \leq 11 \\
& 2x_1-x_2 \leq 16 \\
& -x_1-x_2 \leq -1 \\
& x_2 \leq 5 \\
& x_1, x_2 \geq 0
\end{aligned}$$

Nous allons résoudre ce problème par différentes méthodes, depuis plusieurs points initiaux. Le tableau 4.1 montre les points vers lesquels chacune des méthodes a convergé, en fonction du point initial.

La méthode de barrière a été exécutée avec les paramètres $\epsilon^0 = 10$ et $\beta = 0.8$. Les méthodes ont convergé vers divers minima locaux (notons que tous ces points satisfont la condition nécessaire de Kuhn-Tucker), d'après leur point de départ. La solution optimale globale de ce problème

pt. initial	pt. final Frank et Wolfe	pt. final Zoutendijk	pt. final barrière
(1.0,1.0)	(0.0, 1.0)	(0.0, 1.0)	(0.0, 3.0)
(3.0, 4.0)	(2.0, 5.0)	(2.0, 5.0)	(0.0, 3.0)
(5.0, 2.0)	(9.0, 2.0)	(9.0, 2.0)	(4.2, 5.0)
(4.0, 2.0)	(0.0, 3.0)	(0.0, 3.0)	(4.2, 5.0)
(6.0, 4.0)	(6.0, 5.0)	(6.0, 5.0)	(9.0, 2.0)

TAB. 4.1 – *points finaux des différentes méthodes pour* P_1.

(9,2) n'a été atteinte qu'une fois par chaque méthode. Cela n'est pas surprenant car les g_i sont convexes et f concave. Ainsi, ces trois méthodes, que ce soient les méthodes de directions admissibles qui sont guidées par le gradient ou la méthode de barrière qui utilise la méthode de Newton pour la minimisation directe, sont attirées par les minima locaux situés dans le voisinage de leur points de départ.

Observons plus particulièrement le comportement de la méthode de barrière, pour laquelle nous recommençons l'expérience avec $\epsilon^0 = 50$ (β vaut toujours 0.8), avec les même points initiaux. On constate que, cette fois-ci, elle a convergé à chaque reprise vers (9,2). Cela peut paraître surprenant au premier abord et l'explication en est la suivante : lorsque ϵ est suffisamment grand, la fonction $f(x) + \epsilon B(x)$ devient convexe en dépit de la concavité de $f(x)$. En effet, lorsque ϵ est grand, le terme convexe $\epsilon B(x)$ domine le terme $f(x)$ et augmente plus rapidement que $f(x)$ ne diminue lorsque la frontière du domaine est approchée. Ainsi, la méthode directe de Newton converge vers le minimum global de $f(x) + \epsilon B(x)$. Le théorème 4.3.1 s'applique de sorte que la méthode de barrière converge vers le minimum global de f sur X.

Après que l'exécution ait démarré avec un ϵ^0 suffisamment grand, ce paramètre est progressivement réduit, et ainsi le terme $f(x)$ reprend petit à petit le dessus sur le terme $\epsilon B(x)$. $f(x) + \epsilon B(x)$ n'est donc pas convexe, et il se pourrait très bien que la méthode de Newton converge vers le maximum global (4.2, 1.9). Cependant, encore faudrait-il qu'elle soit démarrée à proximité suffisante de ce point. La méthode de barrière utilise toujours le point obtenu à l'itération précédente comme point de départ de la prochaine exécution de la méthode de Newton. Or, ce point sera trop proche du minimum recherché et trop loin de ce maximum, dû aux itérations précédentes qui ont permis de s'en rapprocher. Lorsque ϵ est faible et que $f(x)+\epsilon B(x)$ tend vers $f(x)$, la méthode de Newton est lancée depuis un point proche du minimum recherché et donc elle est attirée par celui-ci, en dépit et de la présence d'autres points stationnaire plus éloignés.

Nous avons ainsi pu expérimenter un avantage important de la méthode de pénalité intérieure par rapport aux méthodes de directions admissibles : si les g_i sont convexes et pour peu qu'elle soit démarrée avec des paramètres suffisamment élevés, elle peut converger vers le minimum global d'une fonction f sur un ensemble X même lorsque cette dernière n'est pas convexe, contrairement aux méthodes de directions admissibles qui fatalement convergeront toujours vers des minima locaux. Touefois, le prix à payer pour cela réside dans une convergence lente lorsque les ϵ^k sont grands.

Remarque importante : le raisonnement ci-dessus est uniquement valable si la fonction de barrière inverse est utilisée. En effet, avec l'emploi de la barrière logarithmique, $B(x)$ peut être négative car la fonction -ln tend certes vers l'infini lorsque son argument approche 0, mais est également négative si son argument est supérieur à 1.

Cette expérience a aussi permis d'entrevoir des désavantages notables de cette méthode par rapport aux autres : elle ne converge pas de manière finie et a besoin de beaucoup plus d'ité-

rations : les méthodes de directions admissibles n'ont terminé qu'après une ou deux itérations, alors que la méthode de barrière en a nécessité beaucoup plus, de l'ordre de plusieurs dizaines, pour peu que l'on ne souhaite pas une approximation trop précise (cela dépend des $\{\epsilon_k\}$). Les itérations de la méthode de barrière sont également plus gourmandes en temps de calcul. Il faut recommencer la méthode de barrière plusieurs fois à cause du choix des paramètres. Enfin, celle-ci nécessite un point initial intérieur alors que les autres se contentent d'un point initial admissible.

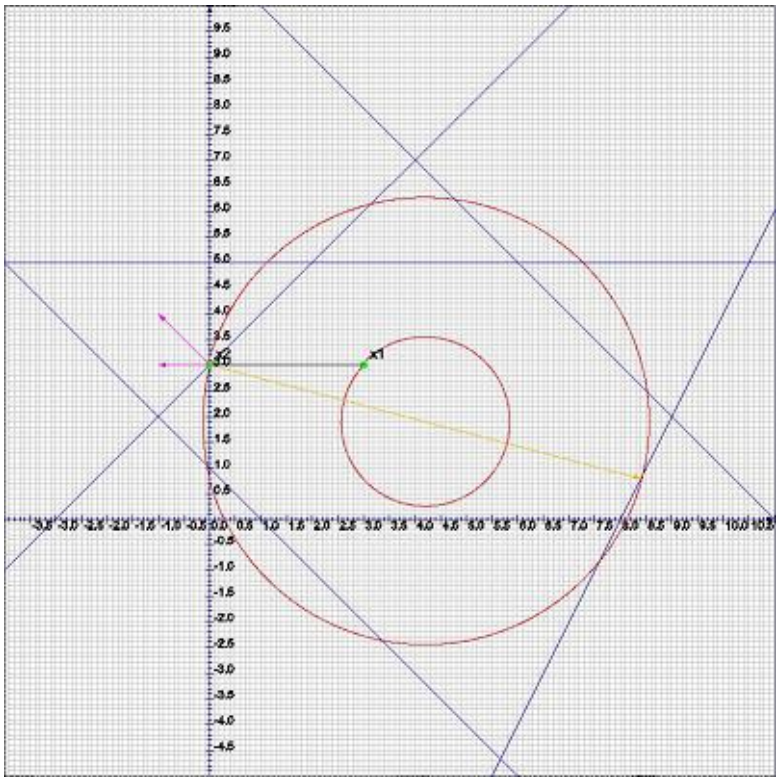

FIG. 4.1 – *illustration la convergence en une itération de la méthode de Zoutendijk vers un point satisfaisant les conditions de Kuhn-Tucker pour le problème P_1 (sur la figure, le gradient de f est représenté en jaune au point optimal (0,3) et les gradient des contraintes actives sont en violet), à partir du point (3,3).*

Considérons le problème suivant P_2 :

$$\begin{aligned}
&\text{minimiser} - (x_1 - 20)^2 - (x_2 - 10)^2 \\
&\text{sous contraintes} - x_1 + x_2 \leq 3 \\
&\qquad\qquad x_2 - \frac{1}{2}x_1 \leq 10 \\
&\qquad\qquad (x_2 - 10)^2 + x_1^2 \leq 500 \\
&\qquad\qquad x_1, x_2 \geq 0
\end{aligned}$$

Résolvons ce problème par la méthode de pénalité extérieure, en démarrant au point non admissible (0, -30), avec un paramètre $\mu^0 = 0.5$ et un facteur d'augmentation de 1.5. La méthode

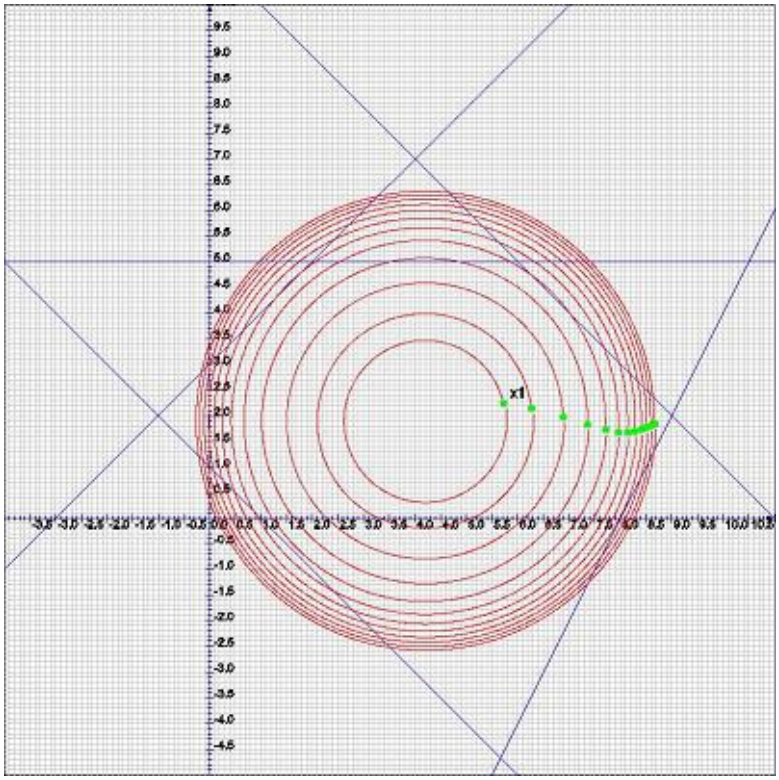

FIG. 4.2 – *illustration la convergence de la méthode de barrière vers le minimum global (9,2) en partant du même point avec des paramètres suffisamment élevés ($\epsilon^0 = 40$ et $\beta = 0.7$) pour P_1.*

converge vers (0, 0), qui est la solution optimale globale. Il est intéressant de noter la "trajectoire" des itérations, visible sur la figure 4.3.

Celle-ci longe d'abord la frontière de la contrainte $(x_2-10)^2+x_1^2 \leq 500$. Ceci peut s'expliquer de la manière suivante : le point x_1 obtenu après la première minimisation se trouve pratiquement sur la frontière de cette dernière contrainte, la pénalité associée est donc quasiment nulle. La méthode cherche la manière la plus directe de minimiser $f(x) + \mu P(x)$, c'est pourquoi elle tente de minimiser la pénalité associée à la contrainte linéaire voisine $-x_1 + x_2 \leq 3$, avant de se déplacer plus directement vers la région admissible.

En recommençant la méthode depuis d'autres point initiaux tels que (40, 40), (300, -200) ou encore (-3000, 2500) la séquence de points obtenus est semblable à la précédente, ce qui tend à montrer que la méthode n'est pas sensible au point initial.

On utilise à présent la méthode de Zoutendijk pour le cas des contraintes non linéaires. Si cette méthode est lancée depuis le point initial admissible (15, 15), on observe qu'elle est attirée par le minimum local (13.5, 16.5). Lancée depuis (20, 20), elle termine au minimum local (19, 22). Lancée depuis (10, 5), elle converge vers le minimum global (0, 0). On remarque que cette méthode est également sujette à une forme zigzag : lancée depuis un point admissible quelconque, elle se déplace immédiatement en un point où une ou plusieurs contraintes sont actives. Dû à la condition supplémentaire que nous avons ajoutée afin de traiter les contraintes non linéaires (cette condition impose l'inégalité stricte de la relation $\nabla g_i(x^k) \cdot d^k < 0$, où g_i est une contrainte active en x^k), lorsque la méthode se trouve en un point où une contrainte, linéaire cette fois, est active, il lui est impossible de se déplacer le long de sa frontière, puisque la direction de

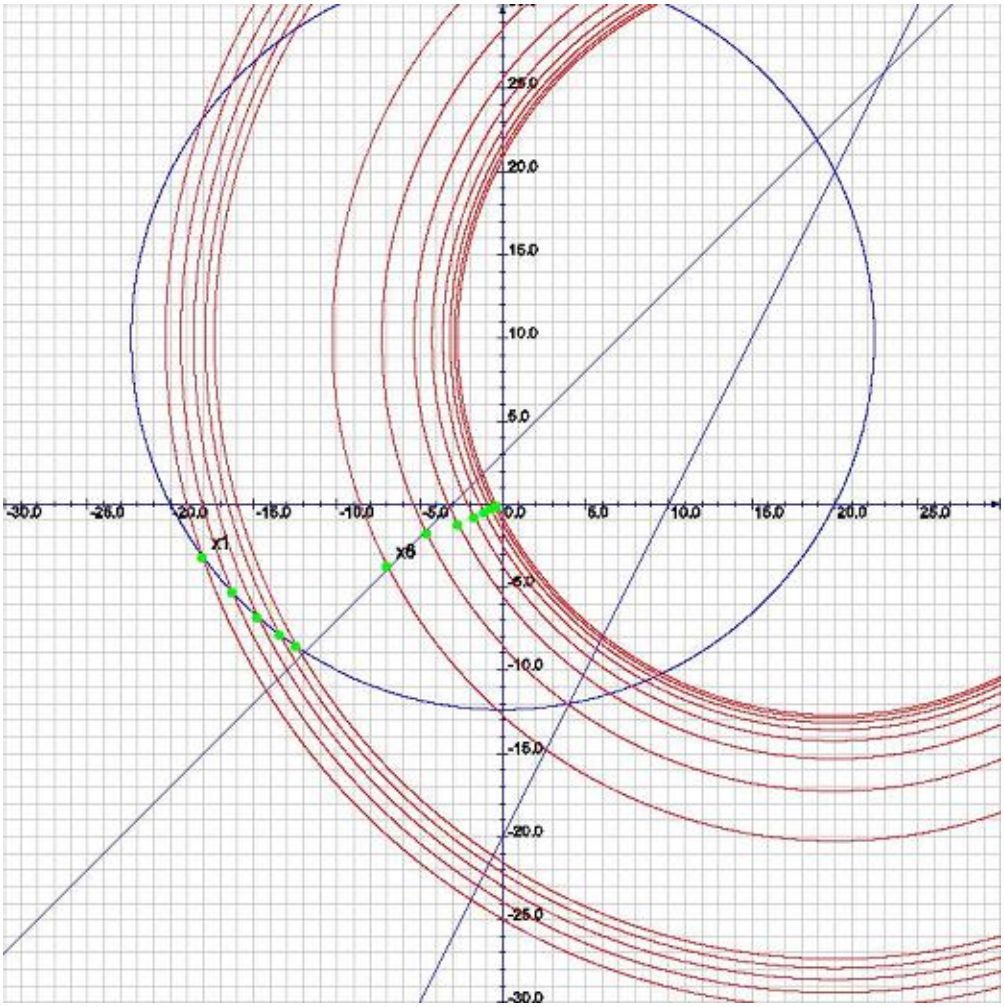

FIG. 4.3 – *illustration de la convergence de la méthode de pénalité extérieure pour le problème* P_2.

mouvement choisie ne peut être orthogonale au gradient de cette dernière. Dans un tel cas, elle se déplace donc en direction d'un point où une autre contrainte est active, donnant lieu au zigzag. Le prix pour le traitement des contraintes non linéaires est ainsi payé par l'obtention d'une convergence nettement plus lente que celle de la méthode pour contraintes linéaires.

Soit P_3 :

$$\begin{aligned} &\text{minimiser } -2x_1 + x_2 \\ &\text{sous contrainte } -x_1^2 - x_2^2 + x_1x_2 + 6x_1 \geq 0 \end{aligned}$$

La fonction objectif est linéaire. Lancée depuis (1, 1), la méthode de Zoutendijk pour les contraintes non linéaires trouve une approximation du minimum global (7.46, 2.0) en 20 itérations. Notons que lancée depuis d'autres points admissibles, elle converge toujours vers ce point, ce à quoi nous nous attendions car f est convexe. Son comportement n'est pas à proprement parler un zigzag, mais consiste à rejoindre un point où la contrainte unique est active et voyage

ensuite par petits déplacements succissifs le long de la frontière, sans traverser celle-ci (cela est visible sur la figure 4.4).

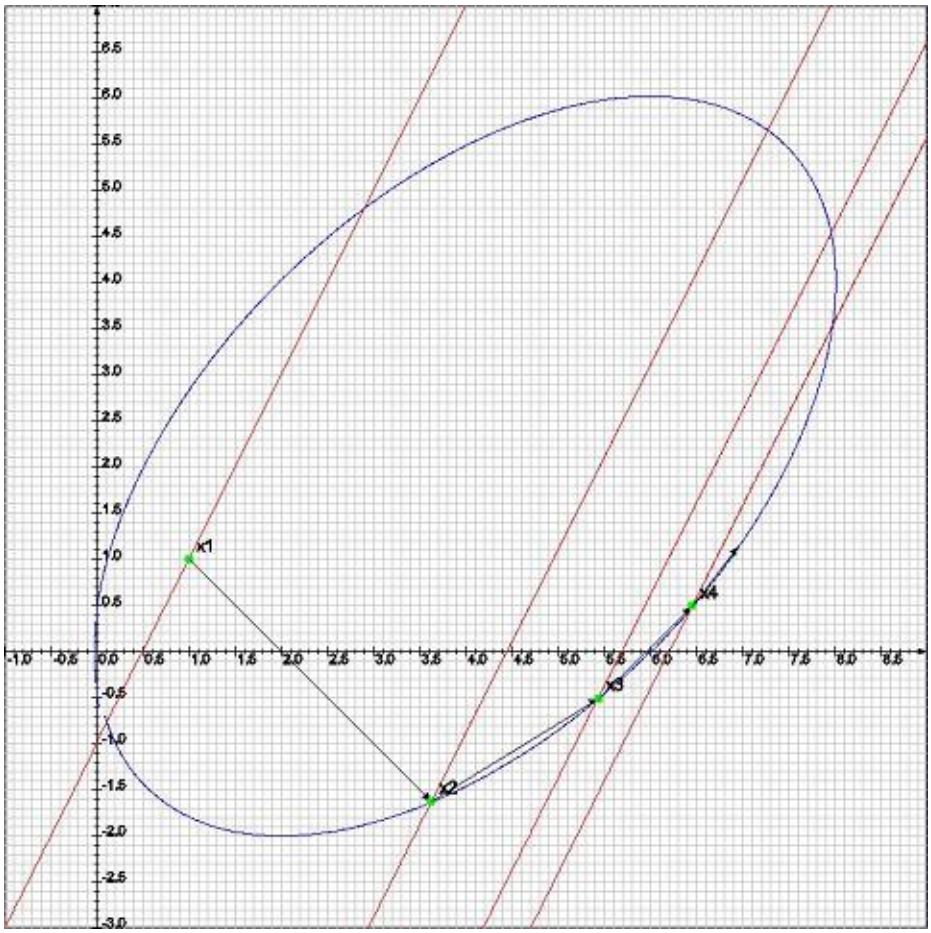

FIG. 4.4 – *illustration des premières itérations de la méthode de Zoutendijk pour P_3 en partant de (1,1).*

On résout maintenant ce problème par la méthode de Kelley, avec les hyperplans initiaux $x_1 \geq 0$, $x_1 \leq 8$, $x_2 \geq -2$ et $x_2 \leq 6$. La méthode termine avec le point (7.46, 2.0) après 19 itérations. Les méthodes de pénalités extérieure et intérieure convergent elles aussi vers le même point, quelque soit le point initial (celui-ci devant tout de même se trouver à respectivement à l'extérieur et à l'intérieur du domaine). On notera tous de même que les méthodes de pénalité sont moins bien adaptées à ce type de problème que les méthodes de Zoutendijk et de Kelley, car elle sont plus gourmandes en temps de calcul, plus lentes, offrent des approximations moins précises et nécessitent le processus ennuyeux de recommencer la méthode plusieurs fois.

Nous considérons maintenant P_4 :

$$
\begin{aligned}
\text{minimiser } & x_1^2 - 2x_2^2 - x_1 - x_2; \\
\text{sous contrainte } & x_1 + 2x_2 - 8 \leq 0 \\
& 3x_1 + x_2 - 9 \leq 0 \\
& x_1 \geq -3 \\
& x_2 \geq -4
\end{aligned}
$$

On remarque que f est une fonction hyperbolique non convexe. Résolvons ce problème par les méthodes de directions admissibles et de pénalité, en commençant par les premières. Avec le point initial (0, 0), nous observons que la méthode de Frank et Wolfe converge en 2 itérations vers le minimum local (-3, 5.5), qui est aussi le minimum global. La méthode de Zouendijk (pour les contraintes linéaires) fait de même en 3 itérations. En partant du point (0,-1), ces deux méthodes convergent en deux itérations vers un autre minimum local (0.5, -4). On recommence depuis (-2, -2) : la méthode de Zoutendijk converge toujours en 2 itérations vers (0.5, -4), mais la méthode de Frank et Wolfe "zigzague" et aura en réalité besoin de 730 itérations pour trouver une approximation de (0.5, -4) avec la condition d'arrêt $|d^k| < 0.005$.

Exécutons maintenant la méthode de barrière pour ce problème, tout d'abord depuis le point (0, 0) avec un paramètre initial $\epsilon^0 = 50$ et un facteur de réduction associé $\beta = 0.9$. Elle converge vers (-3, 5.5). Avec des paramètres $\epsilon^0 = 10$ et $\beta = 0.9$. Nous observons que la méthode converge cette fois vers (0.5, -0.25) qui est le centre de l'hyperbole et un point stationnaire de f. Un raisonnement analogue à celui-ci développé pour le problème P_1 explique cela. Pour les raisons déjà évoquées, si la méthode est lancée depuis n'importe quel point intérieur avec un ϵ^0 et un facteur de réduction suffisamment élevés, nous observons que la méthode converge vers le minimum global.

En recommençant l'expérience depuis (0, -3) avec $\epsilon^0 = 40$ et $\beta = 0.9$, nous constatons qu'alors elle converge vers le minimum local (0.5, -4). En effet, ϵ^0 n'est pas suffisamment élevé et la méthode de Newton étant démarrée à proximité de ce minimum local, elle est attirée par celui-ci.

Réalisons maintenant une série d'expériences avec la méthode de pénalité extérieure. En la lançant depuis n'importe quel point non-admissible avec un paramètre initial faible $\mu^0 = 0.01$, on s'aperçoit qu'en une itération la méthode trouve toujours un point appartenant à l'ensemble admissible et proche de (0.5, -0.25). En effet, μ étant petit, le terme de pénalité est négligeable et donc la méthode de Newton converge vers le seul point stationnaire de f qui est (0.5, -0.25). Un tel choix de paramètre n'est donc pas souhaitable et ne permet pas la résolution du problème.

On relance la méthode avec $\mu^0 = 1.5$ et un facteur d'augmentation $\beta = 1.1$ depuis le point (100, 100) pour constater qu'elle converge vers le minimum global (-3, 5.5). Avec les même paramètres, mais le point initial (100, -100), elle converge à nouveau vers (-3, 5.5).

A présent, nous choisissons $\mu^0 = 2.5$ (toujours avec $\beta = 1.1$). Depuis le point initial (100, 100), la méthode converge à nouveau vers (-3, 5.5). Mais depuis (100, -100), elle converge cette fois vers le minimum local (0.5, -4).

Nous pouvons expliquer cela par le fait qu'avec un paramètre μ et une pénalité plus grande, l'importance de l'admissibilité est accentuée est la méthode cherche à se déplacer en priorité vers un point admissible, même si celui-ci n'est pas le minimum global.

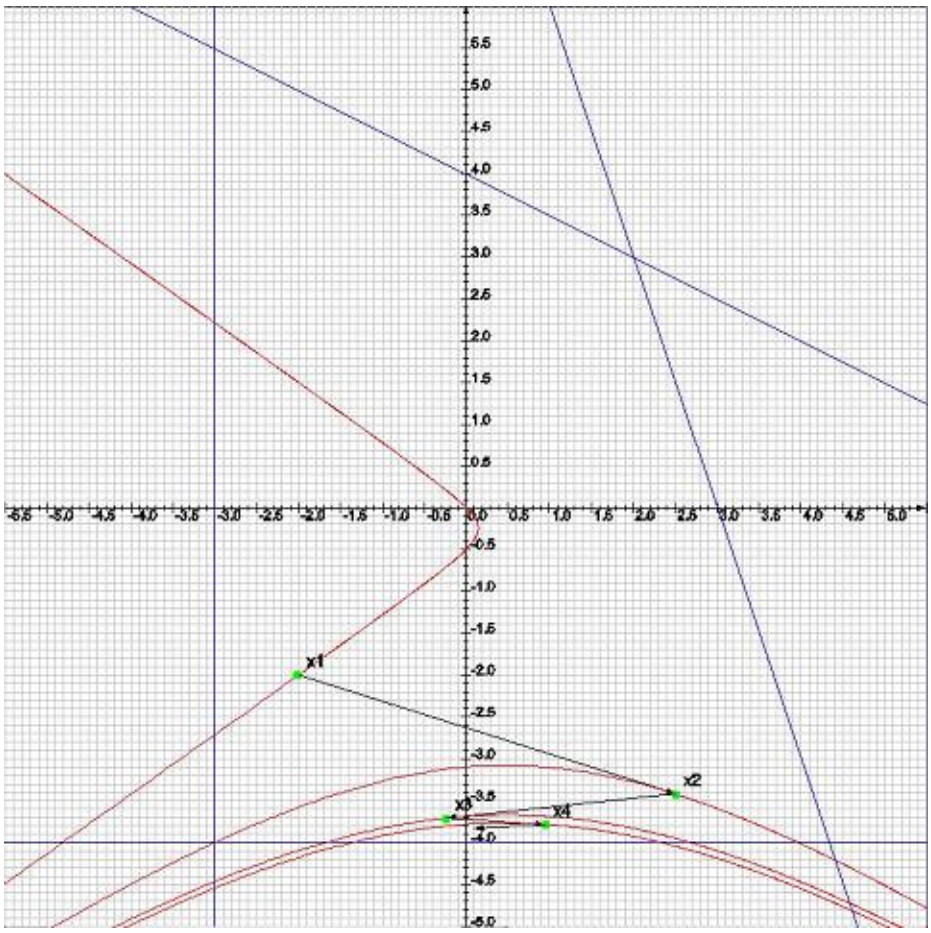

FIG. 4.5 – *illustration du zigzag de la méthode de Frank et Wolfe pour P_4 en partant de (-2,-2).*

Chapitre 5

Manuel d'utilisation de l'application

A partir de ce chapitre, les questions plus spécifiquement relatives au logiciel lui-même sont abordées. On décrit ici la manière dont il doit être utilisé et le chapitre suivant traite de la façon dont il a été implémenté, entrant plus dans les détails de son fonctionnement (le présent chapitre présente la documentation consacrée à l'utilisateur de l'application, alors que le suivant donnera celle destinée au programmeur désireux d'assurer sa maintenance).

5.1 Démarrage de l'application

Cette application a été développée dans le langage Java et nécessite une installation de la machine virtuelle Java (version 1.4.2 ou supérieure) pour fonctionner. Pour l'arithmétique matricielle, elle utilise le paquetage javax.vecmath de l'extension Java3D. Si l'utilisateur dispose d'une installation de Java qui ne comprend pas ce paquetage, deux possibilités s'offrent à lui : installer Java3D (qui peut se trouver sur java.sun.com), ou inclure le fichier vecmath.jar se trouvant sur le CD présent avec ce rapport dans le répertoire adéquat de jre (Java Runtime Environment).

L'application peut être démarrée à l'aide du fichier nonlinearOptimizer.jar présent sur le CD avec la commande suivante :

```
java -jar nonlinearOptimizer.jar
```

5.2 Choix du problème et de l'algorithme pour le résoudre

Fenêtre de l'application

Les éléments de la figure 5.1, qui représente une vue typique de l'application, sont les suivants :

1. Le menu Problème permet de sélectionner le type de problème que l'on souhaite résoudre.
2. Le menu Algorithme permet de choisir un algorithme de résolution en fonction du type de problème choisi.
3. Le menu Dimension permet d'indiquer à l'application la dimension du problème.
4. Le menu Options permet de définir quelques paramètres.
5. Ce champs sert à spécifier la fonction objectif du problème.
6. Ce champs sert à en spécifier les contraintes (visible uniquement pour les problèmes contraints, donc).

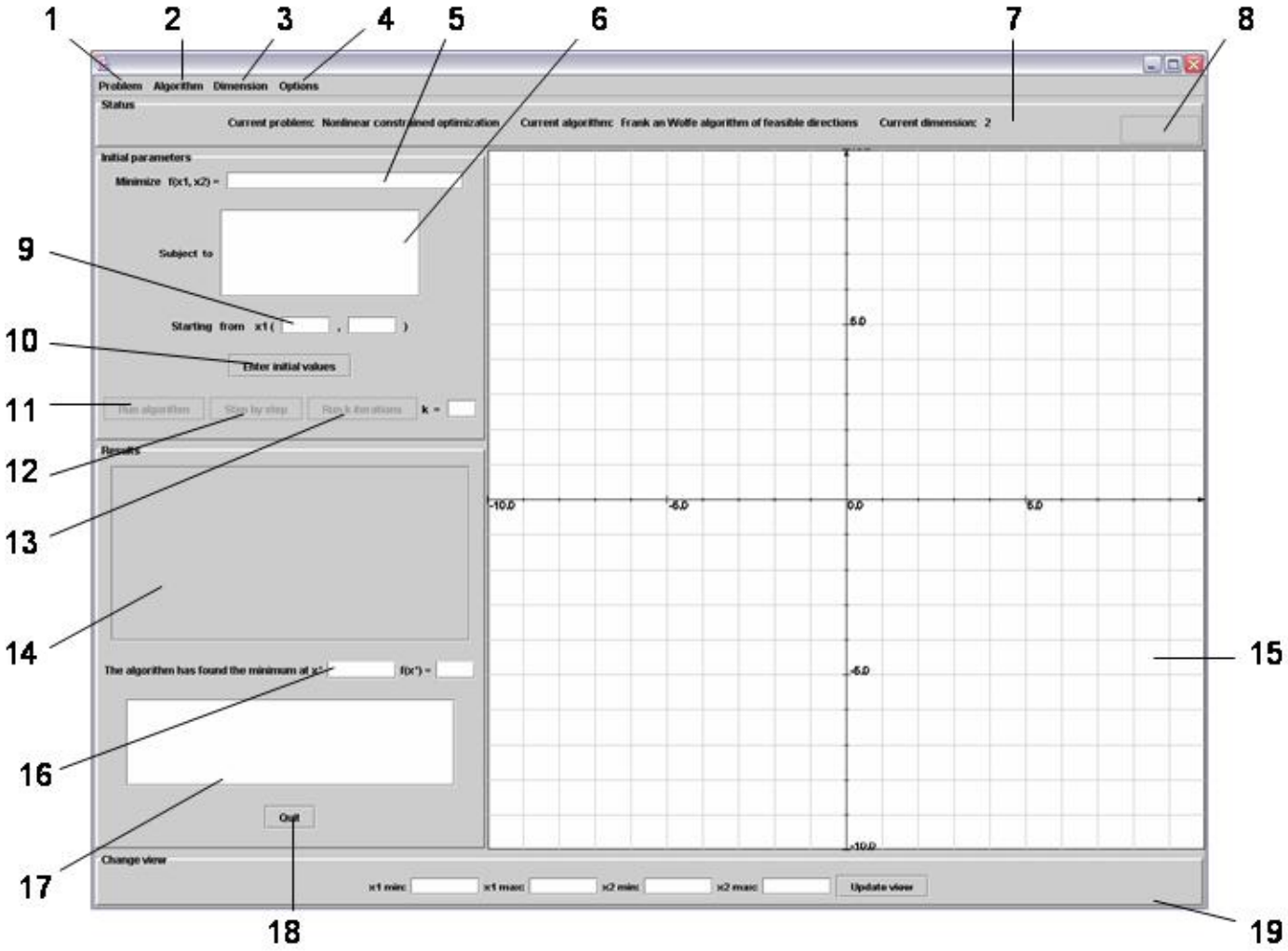

FIG. 5.1 – *la fenêtre principale de l'application.*

7. Cette barre affiche le statut de l'application, soit le type de problème, l'algorithme et la dimension courants.
8. Ce champ permet d'afficher les coordonnées d'un point de la zone d'affichage graphique, lorsqu'on positionne le curseur sur celui-ci.
9. Ces champs servent à préciser le point de départ de l'algorithme.
10. Ce bouton valide les paramètres initiaux une fois ceux-ci entrés.
11. Ce bouton déclenche l'exécution d'un algorithme en entier.
12. Ce bouton permet d'exécuter un algorithme en pas à pas.
13. Ce bouton permet de choisir le nombre d'itérations que l'on souhaite exécuter (ce nombre sera donné dans le champs situé à ses côtés).
14. Cette zone contient le tableau qui affiche les valeurs importantes lors d'une exécution.
15. Cette zone est la zone d'affichage graphique permettant d'illustrer une exécution.
16. Ces champs affichent la solution optimale obtenue à la fin d'une exécution.

17. Cette zone affiche les résultats des tests des conditions d'optimalité.
18. Ce bouton permet de quitter l'application.
19. Cette barre permet de changer la zone que l'on souhaite observer avec l'affichage graphique.

Le menu Problème

Il sert à choisir parmi les problèmes d'optimisation non linéaire avec ou sans contraintes.

Remarque : L'application incluant des routines pour la résolution de programmes linéaire (car, comme nous l'avons dit au chapitre 4, certains algorithmes d'optimisation non linéaire exigent la résolution de ce type de sous-problèmes), la possibilité de résoudre directement des programmes linéaires est également offerte.

Le menu Algorithme

L'algorithme de résolution dépend, bien entendu, du type de problème choisi.

Si celui-ci est un problème d'optimisation non linéaire sans contraintes, les algorithmes du gradient, de Newton, des directions conjuguées, du gradient conjugué et du simplexe sont disponibles. S'il s'agit d'un problème contraint, les algorithmes de Frank et Wolfe, de Zoutendijk (les deux versions destinées au cas des contraintes linéaires ou non linéaires), de Kelley, de pénalité intérieure ou de pénalité extérieure peuvent être sélectionnés.

Chaque algorithme ne peut être utilisé que sous certaines hypothèses relative à l'instance du problème considéré (ces hypothèses ont été détaillées aux chapitres 3 et 4).

En cas de résolution d'un programme linéaire, enfin, seul le traditionnel algorithme du simplexe peut être utilisé.

Le menu Dimension

On l'utilise pour indiquer la dimension du problème parmi trois possibilités : 1, 2 et n. Les deux premiers choix activent l'affichage graphique permettant l'illustration des exécutions. L'option n dimensions est prévue pour tout problème de dimension strictement supérieure à 2 ; l'utilisateur sera invité à entrer la véritable valeur de la dimension par la suite. Cette option désactive l'affichage graphique permettant d'illustrer les exécutions.

Si, par hasard, l'utilisateur souhaite résoudre un problème à une ou deux dimensions mais ne souhaite pas utiliser l'affichage graphique (par exemple pour obtenir une exécution plus rapide), il peut procéder en choisissant l'option n dimensions et en fixant ensuite la valeur de n à 1 ou à 2. Tous les algorithmes sont des algorithmes permettant de minimiser des fonctions de $\mathbb{R}^n$ dans $\mathbb{R}$, ils pourraient donc être utilisés pour des problèmes de n'importe quelle dimension. Toutefois, la plupart des méthodes doivent réaliser une recherche monodimentionnelle à chaque itération, qui est assurée par la méthode de Newton. Dans le cas d'un problème à une dimension, les autres algorithmes se borneraient donc à faire appel à la méthode de Newton pour le résoudre, c'est pourquoi seule cette méthode est disponible dans le cas à une dimension.

Le menu Options

Il permet de spécifier divers paramètres. On y trouvera les sous-menus suivants :

Le premier, appelé Précision, autorise l'utilisateur à spécifier deux types d'arrondis (ce que nous appelons ici arrondi est en fait une valeur entière, comprise entre 0 et 15, qui n'est autre que le nombre de décimales auquel l'utilisateur souhaite arrondir une valeur). Le premier est la précision avec laquelle l'utilisateur souhaite voir s'afficher les valeurs résultant d'une l'exécution

(par défaut, elle vaut 2). Quant au deuxième, nous allons maintenant détailler son utilité et en quelles circonstances il est utilisé.

L'application recourra à cet arrondi durant certains calculs lors d'une exécution. Pour comprendre son utilité, considérons l'exemple suivant : imaginons que nous soyons en train de résoudre un problème comportant des contraintes par la méthode de Zoutendijk. Comme cela a été mentionné au cours du chapitre 3, il est possible de montrer que si, en un point x^k, la valeur optimale de la fonction objectif du sous-programme linéaire de recherche d'une direction est égal à zéro, alors x^k satisfait les conditions de Kuhn-Tucker. Cependant, dans la pratique, et dû à la double précision avec laquelle les calculs sont effectués, il arrive rarement qu'une telle valeur soit exactement égale à zéro. Dans le meilleur des cas, le point courant x^k ne sera qu'une approximation située dans le voisinage très proche d'un point vérifiant les conditions de Kuhn-Tucker. Ainsi, la solution opimale en question sera une valeur certes proche mais différente de zéro. La conséquence en est que l'algorithme ne se terminera pas et continuera à itérer, produisant de tous petits déplacement souvent négligeables devant la précision avec laquelle nous souhaitons approcher l'optimum. Pour éviter ainsi des itérations souvent inutiles, ces valeurs peuvent être arrondies, le choix de la précision requise étant laissé à l'utilisateur.

Cet arrondi sera utilisé, plus concrètement, dans les cas suivants :

- Afin de déterminer si les solutions optimales des sous-programmes linéaire sont nulles dans le cas des méthodes de Zoutendijk ou si la norme de d^k est nulle pour la méthode de Frank et Wolfe (cela induisant la terminaison des algorithmes).
- Comme conditions d'arrêt des méthodes de pénalités ; en effet, celles-ci s'arrêtent lorsque les valeurs de la fonction objectif en deux points générés consécutivement x^k et x^{x+1}, arrondies à cette valeur, sont égales.
- Afin de déterminer si une contrainte est active en un point.

Cette valeur vaut 4 par défaut. Nous aurions pu utiliser cette politique en d'autres occasions, par exemple comme condition d'arrêt d'algorithmes de minimisation sans contraintes. Cependant, il paraît dans de tels cas plus approprié de permettre à l'utilisateur de spécifier ces conditions de manière explicite, ainsi que d'éventuellement choisir entre plusieurs conditions d'arrêt.

Ce menu contient une autre option, appelée Interrompre l'exécution. Elle permet de spécifier une valeur entière indiquant quand l'utilisateur souhaite être interrogé pour éventuellement forcer la fin d'une exécution si celle-ci se révèle trop longue. A chaque fois que le nombre d'itérations atteint un multiple de cet valeur, l'application demande si l'exécution doit ou non être poursuivie. Cette valeur est 200 par défaut.

5.3 Spécifications des fonctions et autres données d'entrée d'un algorithme

Fonctions

On s'intéresse ici à la manière de définir les fonctions, qu'il s'agisse de la fonction objectif ou des contraintes. Pour tout problème, les variables doivent être indexées entre 1 et la dimension de celui-ci (c'est-à-dire spécifieés sous la forme x1, x2, ... xn). Les opérateurs reconnus sont les suivants :

Les opérateurs binaires :

- + (addition)

– - (soustraction)
– * (multiplication)
– / (division)
– ^ (puissance)

Les opérateurs unaires :

– - unaire (opposé)
– sin (sinus)
– cos (cosinus)
– tan (tangente)
– exp (exponentielle)
– ln (logarithme naturel)
– sqrt (racine carrée)

Tout autre élément qu'un nombre, des variables sous la forme décrite ci-dessus, les opérateurs précédents ou des parenthèses est prohibé dans une fonction. L'argument des fonctions trigonométriques sera spécifié en radians. L'exponentielle est en base e. Il n'est pas possible de spécifier de variable dans l'exposant d'une puissance. Une fonction objectif doit être terminée par un " ;". Une collection de contraintes doit être donnée en séparant chacune de celles-ci par des "," et la liste achevée par un " ;". Les relations valides pour les contraintes sont "<=", ">=" et "=".

Les relations de priorité entre les différents opérateurs sont celles utilisées usuellement, c'est-à-dire que la multiplication est prioritaire sur l'addition. En l'absence de parenthèses permettant de lever toute ambiguïté sur l'ordre des évaluations à effectuer, les multiplications sont évaluées en premier lieu. La puissance est également prioritaire sur l'addition.

Attention cependant, car il n'y a pas de relation de priorité entre les opérateurs de multiplication et de puissance : ceux-ci seront évalués dans l'ordre où le programme lit la fonction, c'est-à-dire de gauche à droite. Il faut donc prendre garde car si l'on écrit par exemple

```
3*(x1 - 2)^2
```

la multiplication sera la première évaluée et c'est la totalité de l'expression 3*(x1 - 2) qui sera ensuite élevée au carré. Si l'on souhaite évaluer d'abord la puissance, l'emploi de parenthèses est requis :

```
3*((x1 - 2)^2)
```

L'utilisateur est donc invité à utilisé des parenthèses pour éviter l'ambiguïté dans ce cas.

Voici un exemple de la manière de spécifier une fonction objectif pour un problème à deux dimensions :

```
x1^2 - 2*(x2^2) - x1 - x2 ;
```

et un exemple de contraintes :

```
x1 + 5*x2 <= 5,
2*(x1^2) - x2 <= 0,
-x1 <= 0,
-x2 <= 0 ;
```

Vecteurs et matrices

Il arrive fréquemment que l'application nécessite des valeurs d'entrées sous forme de vecteurs ou de matrices, principalement lors de l'exécution d'un algorithme pour un problème de dimension plus grande que 2. Les vecteurs seront simplement donnés entre parenthèses en séparant les élément par des virgules. Par exemple, le vecteur nul de dimension 3 sera écrit

$$(0,\ 0,\ 0)$$

Les matrices quant à elles seront exprimées ligne par ligne, de la plus haute à la plus basse, avec la syntaxe suivante. Par exemple, la matrice identité de dimension 3 sera écrite

$$[(1,\ 0,\ 0),\ (0,\ 1,\ 0),\ (0,\ 0,\ 1)]$$

Remarque : cette même syntaxe est utilisée lors de l'affichage de vecteurs et de matrices par l'application.

Cas nécessitant des données d'entrées sous forme de vecteurs ou de matrices :

- Dans le cas à n dimensions, la syntaxe du vecteur exprimant le point de départ lors de l'exécution de toute méthode sauf de la méthode de Kelley.
- Dans le cas à n dimensions, lors de l'exécution des méthodes des directions ou du gradient conjugué, pour exprimer la matrice Q et le vecteur b qui contribuent à définir la fonction objectif.
- Dans les cas à 2 et à n dimensions, lors de l'exécution de la méthode des directions conjuguées pour exprimer les vecteurs linéairement indépendants requis par la méthode.
- Quelle que soit la dimension, lors de l'execution de la méthode de Nelder et Mead, où le simplexe initial est exprimé sous forme d'une matrice $(n+1) \times n$.

Conditions d'arrêt

Pour les algorithmes sans contraintes, il est généralement possible de sélectionner plusieurs conditions d'arrêt. Un menu déroulant apparaît alors au-dessus des champs destinés au point initial (non visible sur la figure) et permet de sélectionner la condition voulue.

Polytope initial

L'exécution de la méthode de Kelley requiert la spécification d'un polytope initial (voir chapitre 4). Il sera défini dans le champ prévu à cet effet, de manière tout à fait analogue aux contraintes (mais chaque hyperplan sera défini sous forme d'une inégalité, et devra bien entendu être linéaire). Ce champ est situé sous celui consacré aux contraintes (non visible sur la figure).

5.4 Exécution d'un algorithme

Lancement d'une exécution

Une fois le problème, l'algorithme sélectionnés et l'instance du problème précisée, l'exécution peut être démarrée. Les paramètres initiaux sont validés par une pression sur le bouton du même nom (numéro 10 sur la figure 5.1). Trois options sont alors possibles pour l'exécution d'un algorithme, par une pression sur l'un des trois boutons correspondant (11, 12 et 13) :

- L'exécution complète. L'algorithme est exécuté jusqu'à ce qu'une condition d'arrêt soit satisfaite. Toutefois, lorsque le nombre d'itérations atteint la valeur fixée grâce au menu Options, l'utilisateur a la possibilité de l'interrompre. Si l'affichage graphique est activé, il n'est mis à jour que lorsque l'exécution se termine.

– L'exécution en pas à pas. Les itérations sont exécutées une par une et l'affichage est mis à jour après chacune d'entre elles.
– L'exécution d'un nombre fixé d'itérations. Ce nombre est précisé grâce au champ prévu à cet effet. Ces itérations sont exécutées et l'affichage est ensuite mis à jour.

Dans tous les cas, la méthode se termine si une condition d'arrêt est satisfaite.

Réinitialisation d'une méthode

Si, après avoir exécuté quelques itérations d'un algorithme, l'utilisateur souhaite exécuter à nouveau la méthode depuis le début, il peut le faire par une pression sur le bouton de réinitialisation (au même endroit que le bouton d'entrée des valeurs initiales, numéro 10), ou changer de méthode ou de problème en sélectionnant n'importe quelle option dans les menus Problème, Algorithme ou Dimension. Les données résultant de l'exécution précédente sont supprimées.

Affichage des résultats

Tableau de l'exécution A chaque itération, l'utilisateur peut consulter toutes les valeurs intéressantes obtenues durant l'exécution, qui sont affichées dans ce tableau (numéro 14 sur la figure). Il peut s'agir d'éléments variés en fonction de l'algorithme. Il sera question notamment du point courant x^k, de la direction d^k et du point x^{k+1} générés à cette itération, mais également de la valeur du gradient au point x^k, ou encore de la liste des contraintes actives en ce point si le problème est contraint, etc.

Solution optimale Lorsqu'une condition d'arrêt est satisfaite et qu'un algorithme se termine, la solution optimale obtenue et la valeur de la fonction objectif en ce point s'affichent dans les champs prévus à cet effet (numéro 16).

Conditions d'optimalité Lorsqu'une exécution se termine, l'application teste si la solution optimale obtenue vérifie les conditions d'optimalité. Dans le cas sans contraintes, elle teste si la solution satisfait les conditions nécessaire et suffisante du premier et du second ordre. Dans le cas d'un problème avec contraintes, elle teste si le point optimal satisfait la condition nécessaire de Kuhn-Tucker. Dans l'affirmative, elle détermine et affiche la combinaison linéaire du gradient de la fonction objectif et des gradients des contraintes actives au point optimal, montrant que la condition est satisfaite. Les résultats des tests de conditions d'optimalité sont affichés dans l'aire prévue à cet effet (numéro 17).

Remarque : si l'approximation du minimum n'est pas assez précise, il peut arriver que le tests des conditions donnent des résultats erronés.

Fenêtre d'affichage graphique

Cette fenêtre permet de visuliser l'exécutions des algorithmes pour les problèmes en deux ou en une dimension.

Démarrage rapide de l'application

Voici les étapes à suivre pour démarrer et utiliser immédiatement l'application :

1. Démarrer l'application
2. Choisir un problème et un algorithme par le biais des menus Problème, Algorithme et Dimension.

3. Exécuter l'algorithme en utilisant les boutons Exécuter l'algorithme, Pas à pas, ou Exécuter k itérations.
4. En cours ou à la fin de l'exécution, changer de problème ou d'algorithme avec les trois menus cités ci-dessus, ou quitter l'application par le bouton correspondant.

Chapitre 6

Manuel du programmeur de l'application

6.1 Architecture générale

Nous allons maintenant entrer plus dans les détails de l'implémentation de ce logiciel, afin de fournir une documentation qui permette à un développeur qui souhaite maintenir l'application de le faire. Ce chapitre consistera d'abord en la présentation de l'architecture du logiciel, puis en l'exposé des divers paquetages qui le composent.

L'architecture générale de l'application est bâtie selon le concept classique de modèle-vue-contrôleur qui permet de séparer clairement les tâches parmi les différentes branches de l'application : la vue regroupe les routines d'intéraction avec l'utilisateur, le modèle regroupe les méthodes de gestion et de traitement des données et le contôleur prend en charge la gestion des évènements et la synchronisation du modèle et de la vue. On rappelle que le modèle ne connaît rien de ses vues. Lorsque l'utilisateur envoie une requête à l'application, le contrôleur analyse celle-ci, puis demande au modèle approprié d'effectuer les traitements et enfin renvoie la vue adaptée. Cette architecture impose une grande clarté et simplifie la vie du développeur ; en effet, l'un des trois composants peut être librement modifié sans que les autres s'en trouvent affectés. Le lecteur pourra trouver à la section suivante une figure schématisant l'architecture générale de l'application.

Les principales composantes de l'application sont : une classe utilisée pour l'affichage ; toutes les méthodes pour la gestion des entrées et sorties y sont regroupées. Le contrôleur est également implémenté sous forme d'une classe unique. Par contre, une collection de classes ont été implémentées pour le modèle, chaque algorithme d'optimisation se voyant consacrer sa propre classe. Ce choix a été effectué car ces algorithmes reposent parfois sur des principes trop différents les uns des autres ; qui plus est, chaque algorithme est applicables sous des conditions qui lui sont propres. C'est pourquoi il semblait plus aisé de séparer clairement les implémentations de chaque algorithme, plutôt que de les regrouper ou encore de définir une classe générique dont chacune des méthodes serait définie en tant que sous-classe.

On peut noter la présence sur la figure 6.1 du "parser" dont le rôle est de prendre en entrée les fonctions telles que l'utilisateur la définit sous forme d'une chaîne de caractères, et de retourner les fonctions modélisées sous une forme interprétable par les algorithmes ou l'affichage. Il est implémenté sous forme de plusieurs classes, chacune étant consacrée aux différents types de chaînes de caractères à interpréter.

Bien entendu, à ces éléments principaux s'ajoutent un certain nombre de classes annexes implémentant les fonctionnalités additionnelles nécessaires à l'application.

On trouvera ci-dessous une liste des différents paquetages que comprend l'application et de

leur fonction :

- le paquetage **control** contient le contrôleur de l'application.
- le paquetage **display** contient les classes nécessaires à l'interaction avec l'utilisateur, dont la classe principale définissant l'interface graphique.
- le paquetage **method** regroupe les classes implémentant les algorithmes d'optimisation.
- le paquetage **function** contient les classes modélisant les fonctions et les contraintes.
- le paquetage **parser** contient les classes implémentant le parser.
- le paquetage **exception** contient les diverses exceptions qui peuvent être rencontrées et envoyées lors de l'exécution des algorithmes.
- le paquetage **optimalityConditions** contient des classes implémentant les routines permettant le test des conditions d'optimalité.
- le paquetage **linearProgramming** contient les classes regroupant les routines de résolution des programmes linéaires.
- le paquetage **utils** contient diverses classes utilitaires permettant d'exécuter certaines opérations annexes utiles à l'application.

Nous ne donnerons pas ici la liste détaillée des différentes classes du programme et de leur fonctionnement, car cela serait trop long et les détails d'implémentation ne sont pas une chose des plus passionnante. Nous nous contenterons d'expliquer leur fonctionnement global. Pour connaître l'implémentation de manière très détaillée, le lecteur pourra se reporter à la Javadoc qui décrit le rôle et le fonctionnement de chaque classe et méthode et qui est présente sur le CD dans le répertoire programme/javadoc.

6.2 Fonctionnement de l'application

Afin d'assurer un fonctionnement correct et de certifier que, en tout instant, l'algorithme convenable est exécuté pour le problème adéquat et suivant les bons paramètre initiaux, il est nécessaire de régir l'exécution de l'application par une machine d'états. Celle-ci permet au différentes partie de l'application de déterminer en tout temps quel type de problème est en train d'être résolu, par quel algorithme et quelle est sa dimension, ce qui est primordial. En effet, l'affichage par exemple devra évidemment être différent si le problème est en 2 dimensions ou d'une dimension supérieure. Lorsque l'utilisateur souhaite exécuter les itérations d'un algorithme, il est nécessaire que le contrôleur "sache" toujours de quel algorithme il s'agit, etc. Cette machine est simple et comporte en réalité autant d'états que de combinaisons possibles du type de problème, de l'algorithme et de la dimension (celle-ci pouvant être 1, 2 ou n). Elle est implémentée dans le contrôleur. Le graphe représentant cette machine contient un grand nombre de noeuds et d'arc, nous renonçons donc à le présenter ici. Lors du démarrage de l'application, celle-ci se trouve dans l'état par défaut suivant : le type de problème est l'optimisation sans contraintes, la méthode est celle du gradient et la dimension est égale à 2. Puis l'état varie en fonction des choix de l'utilisateur.

Venons-en au fonctionnement lors d'une exécution typique. Le diagramme de collaboration de la figure 6.1 illustre l'intéraction entre les différentes parties de l'application.

Les étapes visibles sur ce diagramme sont les suivantes :

1. L'affichage appelle une méthode du contrôleur indiquant le choix de l'utilisateur quant au problème, à l'algorithme et la dimension. Celui-ci met son état à jour.
2. Le contrôleur met ensuite à jour l'affichage, en fonction de son état.
3. Lorsque l'évènement correspondant est détecté, l'affichage invoque les méthodes du parser afin de modéliser la fonction objectif (et éventuellement les contraintes) que l'utilisateur

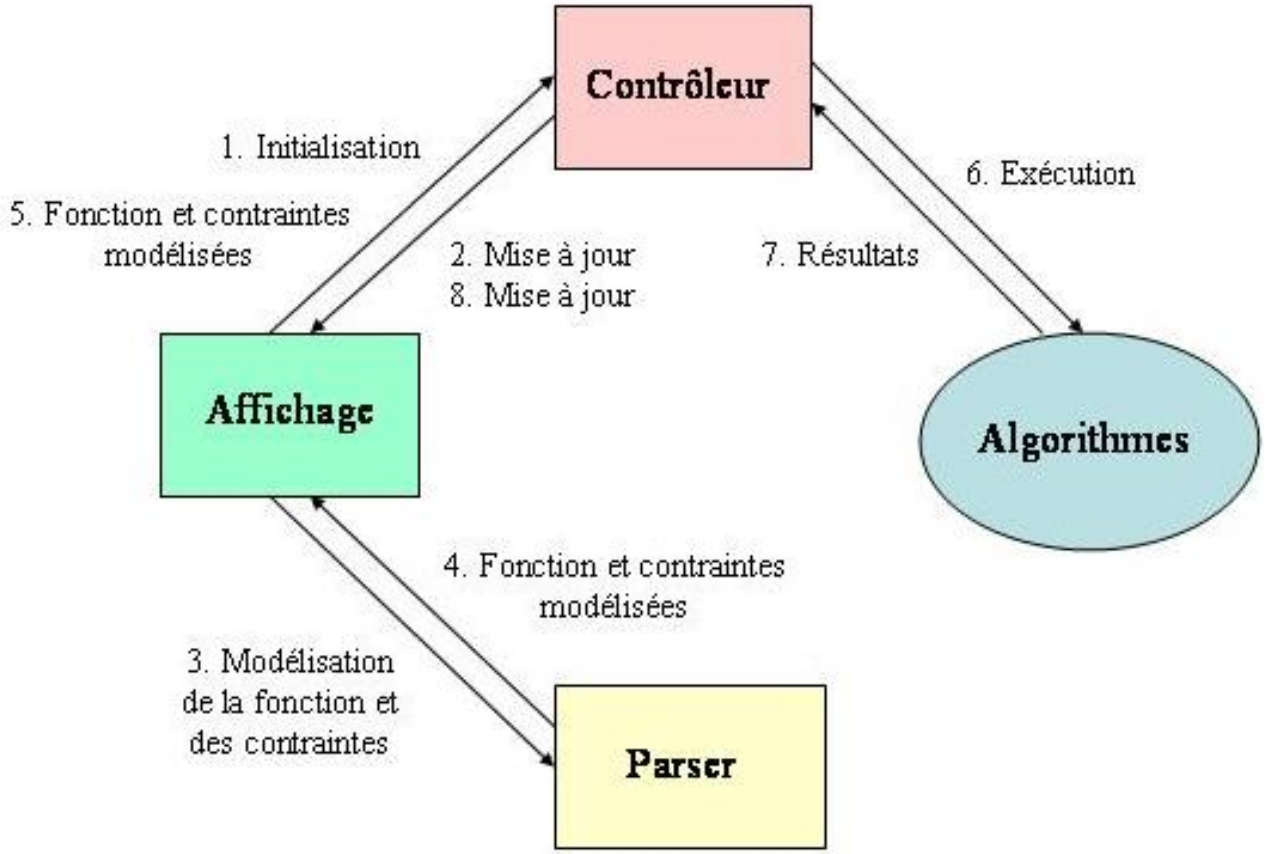

FIG. 6.1 – *diagramme de collaboration des différents éléments de l'application.*

vient d'entrer.

4. La fonction et les contraintes modélisées sont transmises en retour à l'affichage.
5. L'affichage transmet au contrôleur la fonction et les contraintes modélisées ainsi que les autres valeurs initiales nécessaires. Ce dernier est à présent en mesure d'initialiser le bon algorithme avec les valeurs initiales correctes.
6. En fonction des évènement provoqués par l'utilisateur, l'interface transmet au contrôleur le nombre d'itérations à effectuer. Celui-ci appelle de manière adéquate les méthodes de la classe implémentant l'algorithme.
7. Une fois les itérations requises exécutées, l'algorithme transmet les résultats au contrôleur qui met la vue à jour. Ces deux dernières étapes (6 et 7) peuvent être répétées un nombre quelconque de fois.
8. Finalement, lorsqu'une condition d'arrêt est satisfaite à la fin d'une itération, l'algorithme teste les conditions d'optimalité et transmet tous les résultats au contrôleur qui met à jour la vue.

Il est à noter que, pour peu qu'une exécution complète de la méthode n'ait pas été ordonnée, celle-ci peut être interrompue à tout moment si l'utilisateur choisi de démarrer une nouvelle exécution, où change l'un des éléments tels que cette dernière, le problème ou la dimension. Le contrôleur passe alors dans l'état correspondant et les données relatives à l'exécution précédente sont supprimées.

6.3 Représentation des fonctions

La représentation des fonctions que nécessite une telle application doit être en mesure de répondre à de nombreuses requêtes. Bien entendu, nous devons être capables d'évaluer celles-ci ainsi que de calculer leurs dérivées. Il est également souhaitable de pouvoir, par exemple, vérifier si une fonction est linéaire, si oui déterminer les coefficients des différentes variables, vérifier qu'une fonction contienne des variables, ou encore nous assurer qu'elle soit conforme à certaines restrictions assurant le fonctionnement correct d'un algorithme.

Une élégante solution répondant à ces exigences consiste en la représentation des fonctions sous forme d'arbre binaires, où les feuilles sont constituées des valeurs (constantes et variables) et où les noeuds internes sont les opérateurs. Du point de vue de la programmation, il faut que la structure de ces arbres soit telle que tout arbre se compose d'un noeud, d'un sous-arbre gauche et d'un sous-arbre droit (et non pas, par exemple, d'un ensemble de noeuds). Grâce à cette représentation, la récursivité peut-être utilisée afin d'effectuer aisément et efficacement toutes les opérations nécessaires sur les fonctions : évaluation, calcul des dérivées première et seconde, tests déterminant si la fonction est linéaire, si elle est conforme aux restrictions, etc.

Pour intégrer les opérateurs unaires à un arbre, il suffit de laisser dans ce cas l'un de ses deux sous-arbres vides. Ceux-ci sont construits par le parser directement d'après les fonctions tapées au clavier par l'utilisateur.

6.4 Eléments au sujet de l'affichage graphique

Afin d'illustrer le fonctionnement des algorithmes, il est nécessaire de représenter les courbes planes de fonctions de deux variables données implicitement $f(x_1, x_2) = 0$. Il existe de nombreuses solutions pour cela. L'une d'elles consiste à "quadriller" le plan et tester si, en l'ensemble des points obtenus la valeur de f est égale, strictement supérieure, ou strictement inférieure à zéro. Une recherche dichotomique peut alors être effectuée entre deux points situés de part et d'autre de la courbe, jusqu'à obtenir une approximation d'un point de celle-ci. En répétant cette opération, on peut obtenir un grand nombre de points de la courbe qu'il suffit ensuite de relier (dans le bon ordre) pour dessiner celle-ci. L'inconvénient et que si nous voulons obtenir des points suffisamment proche, nous devons évaluer la fonction de nombreuses fois. Cela oblige, de plus, à ordonner les points de la courbe avant de la dessiner.

Une approche alternative, qui a été utilisée pour cette application, consiste à utiliser une approximation linéaire de la courbe, qui n'est autre que le gradient de la fonction. Il est possible d'utiliser cette approximation pour obtenir des vecteurs tangents à la courbe permettant de tracer des segments qui, s'ils sont suffisamment petits, forment une approximation tout à fait convenable de la courbe elle-même. Cela évite le processus précédent, plus coûteux en temps de calcul. On peut s'arranger pour qu'un seul point de la courbe soit nécessaire pour dessiner celle-ci, mais l'inconvénient de cette approche est la manque de précision quelle induit en certaines circonstance, nous obligeant à recalculer les points lors, par exemple, d'un agrandissement de la vue.

Chapitre 7

Conclusion

Cette conclusion sera, je l'espère, assez courte, car le présent document est, lui, déjà suffisemment long. Ce projet a été très intéressant car il m'a donné l'opportunité de travailler dans des domaines variés, me permettant de concilier un travail théorique avec une tâche pratique d'implémentation. Il m'a donc permis d'améliorer grandement mes connaissances à la fois en mathématiques, dans le domaine de l'optimisation non linéaire, et en programmation. Il m'a permis d'apprécier par moi-même l'efficacité des divers types de méthodes, de juger de leur convenance envers les différents types de problèmes, d'être confronté aux difficultés qui surviennent lors du passage entre les principes théoriques sur lesquels les méthodes sont fondées et leur implémentation à l'aide d'un programme réel.

Ce sujet passionnant n'a bien sûr pas été épuisé à travers ce projet et de nombreux défis restent ouverts, en particulier l'implémentation de nouvelles méthodes et d'extensions aux modèles présentés, afin de juger des gains en terme de performance qui pourraient en résulter, ou de tenter d'élargir leurs champs d'application.

En guise de conlusion, je dirai donc que je suis heureux d'avoir effectué ce projet qui m'aura permis à la fois de me pencher sur des concepts mathématiques fondamentaux et également de m'atteler à une tâche de conception d'un logiciel complet.

Références

[1] J. Abadie, *Nonlinear programming*, North Holland, 1967.

[2] American mathematical Society, *Nonlinear programming*, Library of Congress Cataloging in Publication Data, 1976.

[3] Mordecai Avriel, *Nonlinear programming : Analysis and Methods*, Prentice-Hall, 1976.

[4] Mokhtar S. Bazaraa, C.M. Shetty, *Nonlinear programming : Theory and Algorithms*, John Wiley and Sons, 1979.

[5] Dimitri P. Bertsekas, *Nonlinear programming*, Athena Scientific, 1995.

[6] P.E. Gill, W. Murray, *Numerical methods for constrained optimization*, Academic Press, 1974.

[7] Reiner Horst, Panos M. Pardalos, Nguyen V. Thoai, *Introduction to global optimization*, Kluwer Academic Publishers, 2000.

[8] Reiner Horst, Hoang Tuy, *Global Optimization : Deterministic Approaches*, Springer-Verlag, 1990.

[9] J.E. Kelley, *The cutting plane method for solving convex programs*, J. Soc. Ind. Appl. Math. vol.8 p.703-712, 1960.

[10] H.W. Kuhn, A.W. Tucker, *Nonlinear programming*, dans *Proceedings of the second Berkeley Symposium on Methematical Statistics and Probability* (J. Neyman), Uneversity of California Press, 1951.

[11] F.A. Lootsma, *Numerical Methods for Non-linear Optimization*, Academic Press, 1971.

[12] F.A. Lootsma, *Hessian matrices of penalty functions for solving constrained optimization problems*, Philips Res. Rept vol. 24 p.322-331, 1969.

[13] F.A. Lootsma, *Boundary properties of penalty function for constrained minimization*, Philips Res. Rept Supp. vol. 3, 1971.

[14] F.A. Lootsma, *Numerical Methods for Non-linear Optimization*, Academic Press, 1971.

[15] O.L. Mangasarian, R.R. Meyer, S.M. Robinson, *Nonlinear programming*, volume 1, 2, 3, Academic Press, 1972, 1975, 1978.

[16] G.P. McCormick, *A second order method for the linearly constrained programming problem*, dans [15] vol.1 p.207-243, Academic Press, 1970.

[17] Michel Minoux, *Programmation mathématique : théorie et algorithmes*, Dunod, 1989.

[18] J.A. Nelder, R. Mead *A simplex method for function minimization*, Computer J. vol.7 p.308-313, 1965.

[19] G.L. Nemhauser, A.H.G. Rinnooy Kan, M.J. Todd, *Optimization*, volume 1, North Holland, 1989.

[20] David A. Wismer, R. Chattergy, *Introduction to nonlinear optimization : a problem solving approach*, North Holland, 1978.

[21] R. Wolfe, *On the convergence of gradient methods under constraints*, Research Rept. IBM Zurich Laboratory, 1966.

[22] G. Zoutendijk, *Methods of feasible directions*, Elsevier, 1960.

[23] G. Zoutendijk, *Mathematical Programming Methods*, North Holland, 1976.